
Universidade de Aveiro Departamento de Matemática
2008

Alexandre Leite de Castro Madeira

Abordagem Algébrica à Igualdade Observacional


Universidade de Aveiro Departamento de Matemática
2008

Alexandre Leite de Castro Madeira

Abordagem Algébrica à Igualdade Observacional

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Matemática, realizada sob a orientação científica do Doutor Manuel António Gonçalves Martins e do Doutor Luís António Arsénio Descalço, Professores auxiliares do Departamento de Matemática da Universidade de Aveiro.

Dedicado este trabalho à Susana

o júri

Presidente

Prof. Doutor Domingos Moreira Cardoso
Professor Catedrático do Departamento de Matemática da Universidade de Aveiro

Prof. Doutor Luís Manuel Dias Coelho Soares Barbosa
Professor Associado da Escola de Engenharia da Universidade do Minho

Prof. Doutor Manuel António Gonçalves Martins
Professor Auxiliar do Departamento de Matemática da Universidade de Aveiro

Prof. Doutor Luís António Arsénio Descalço
Professor Auxiliar do Departamento de Matemática da Universidade de Aveiro

agradecimentos

-Ao meu orientador Prof. Doutor Manuel Martins, pela incansável presença em todo este percurso. Foi um privilégio e um desafio trabalhar sob tão generosa, encorajadora e rigorosa orientação.

-Ao meu co-orientador Prof. Doutor Luís Descalço por todas as sugestões e comentários.

-Ao Prof. Doutor Enrique Hernandez Manfredini por todo o interesse e, principalmente, por todo o apoio na preparação do artigo ``Observational Refinement Process''.

-À Prof. Doutora Rosália Rodrigues e ao Prof. Doutor Dirk Hoffman pela confiança, nomeadamente nos seu apoios à minha candidatura a Doutoramento. Registe-se também, neste mesmo sentido, o meu agradecimento ao Prof. Doutor Luís Soares Barbosa.

-Aos participantes do recém formado Grupo-Lógica, em especial, às minhas colegas Jacinta Poças e Nilde Barreto. Penso que foi muito enriquecedor para este trabalho, ter tido a oportunidade de ouvir, apresentar e discutir assuntos tão interessantes, num ambiente de grande camaradagem.

-Ao Prof. Doutor Rolf Hennicker pelos esclarecimentos e sugestões que gentilmente me prestou. Ao Prof. Doutor Donald Sannella, por me conceder o seu livro “Foudations of Algebraic Specification and Formal Program Development” ainda em fase de rascunho.

-Aos meus pais, irmãos, sobrinhas e amigos, por todo o apoio, amizade e carinho.

-À minha namorada por tudo...

A. M.

palavras-chave

Especificação algébrica de sistemas de software, igualdade observacional, igualdade comportamental, equivalência comportamental, equivalência observacional, processo de refinamento passo-a-passo.

resumo

A *especificação algébrica de sistemas de software* é um importante tópico dos denominados *métodos formais de desenvolvimento de software*. Neste conte*xto, modelam-se* progra*mas por álg*ebr*as e as suas computaçõe*s por *termos, recorren*do-se a*os resultados da Álgebra Universal e da Lógica,* como ferramentas de verificação e apoio ao processo de implementação. Em grande parte *dos trabalhos sobre o t*ema presentes na literatura, usa-se a *Lógica Equacional* como lógica de suporte a estes processos. Contudo, esta lógica mostra-se limitada para a especificação de programas *Orientados a Objectos*, nomeadamente na especificação de programas com dados encapsulados. A separação entre os aspectos internos e externos do si*stema* induz *uma nova perspectiva do* conceito de modelação, segundo a qual, um objecto se considera como sendo uma realização *correcta do sistema, se satisfaz o*s seus requisitos observacionalmente, isto é, se os resultados *das computações sobre si* executadas satisfazem esses requisitos, podendo não os satisfazer em sentido estrito. Seguindo esta linha de ideias, dois objectos de software são considerados equivalentes quando se comportam da mesma forma perante todas as possíveis computações. Este paradigma é denominado por *Abordagem Observacional de Sistemas*. Uma forma de adequar a *Lógica Equacional* a esta abordagem, é pela substituição da igualdade estrita pela relação de *Igualdade Observacional*, segundo a qual dois elementos se consideram iguais quando se comportam da mesma forma perante qualquer computação, isto é, se produzem os mesmos outputs perante as mesmas computações.

Neste trabalho estuda-se a abordagem observacional de sistemas segundo diferentes grupos de investigação, com especial atenção aos trabalhos da *Lógica Escondida* (por *Goguen-Rosu*), *Lógica Comportamental e Observacional* (por *Bidoit-Hennicker*) e da *Lógica Algébrica* (por *Pigozzi-Martins*). Um ponto central do texto é a generalização do processo de desenvolvimento de software por *Refinamento Passo-a-Passo* a este paradigma. Aprofundam-se aqui algumas variantes deste tópico, incluindo o caso onde se admitem encapsulamentos e desencapsulamentos de dados durante o processo de refinamento.

Numa primeira fase do texto o assunto é apresentado ao nível mais geral das *especificações algébricas estruturadas* (e não exclusivamente do caso das especificações flat) e das *igualdades comportamentais* (congruências parciais arbitrárias).

keywords

Algebraic specification of software systems, observational equality; behavioural equality; observational equivalence; behavioural equivalence; stepwise refinement process;

abstract

The *algebraic specification of software systems* is an important topic of so-called *formal methods of software development*. In this context, programmes are modelled by algebras and computations executed over them by terms, using up the results from *Universal Algebra* and *Logic*, as verification and support tools for the implementation process. In a large majority of the works about this subject, it uses the *Equational Logic* as support logic for these processes. However, this logic is too restrictive for the specification of *object-oriented programs*, namely, in the programs specification with encapsulated data. The split between the internal and external aspects of the system, induces a new perspective of the modelling concept, whereby an object is considered a correct realization of the system if satisfies observationally their requirements, that is, if the results of computations over it executed satisfies these requirements and being able not to satisfy them in the strict sense. Following this principle, two software objects are considered equivalent when behave the same way at all possible computations. This paradigm is called *Observational Approach of Systems*. One way to adjust the *Equational Logic* to the observational approach is by replacing the strict equality by the relation of *Observational Equality*, according to which two elements are considered equal when behave the same way at the same computations, i.e., if they produce the same outputs before the same computations.

We follow this approach according to different research groups, with special attention to the work of *Behavioural* and *Observational Logic* (by *Bidoit-Hennicker*), the *Hidden Logic* (by *Goguen-Rosu*) and *Abstract Algebraic Logic* (by *Pigozzi-Martins*). A central point of the text is the generalization of the software development process by *stepwise refinement* to this paradigm. Here some variants of this topic are explored including the case where encapsulated and desencapsulated data are allowed during the refinement process.

In a first stage of the text, the subject is presented to a more general level of *structured specifications* (and not exclusively the case of *flat* specifications) and the *Behavioural Equalities* (arbitrary partial congruence).

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

A complexidade dos sistemas de software da actualidade requer o estudo de metodologias rigorosas que permitam um desenvolvimento de programas organizado, consistente e sustentável. É neste contexto que se estuda a *especificação formal de sistemas de software*. De forma intuitiva, o processo de implementação de software pode ser entendido como a construção de uma sequência de descrições, cada vez mais precisas, do comportamento do sistema pretendido. Neste processo, parte-se de uma descrição do "que o sistema deve fazer"e chega-se à descrição de "como o deve fazer", isto é, à descrição da implementação procurada. Por forma a evitar as ambiguidades inerentes à linguagem natural, devem estar definidos, para este processo, uma *sintaxe* e um *sistema de inferência*, por forma a ser possível descrever propriedades e verificar a admissibilidade dos passos do processo e uma *semântica associada*, isto é, a descrição de uma classe de objectos matemáticos, denominados de *modelos*, que satisfaçam as propriedades da especificação. São estas as três componentes oferecidas pelas denominadas *linguagens de especificação formal* (cf. [GB99, BH08]). Estas linguagens dividem-se em dois grandes grupos: as linguagens *orientadas a modelos* e as linguagens *orientadas às propriedades*, também conhecidas por linguagens *de especificação algébrica*. Segundo o primeiro grupo, o processo de especificação traduz-se na construção de um único modelo para realizar o sistema, feita a partir de um conjunto de operadores primitivos definidos nas próprias linguagens de especificação. Segundo a *especificação algébrica*, programas são vistos como álgebras e as computações sobre eles executadas como termos. Neste contexto, quando se pretende implementar um determinado sistema de software, procede-se da seguinte forma: define-se uma *assinatura* adequada ao sistema a implementar, tendo em conta as suas funções e tipos de dados e, de seguida,

expressam-se as propriedades desejadas destas operações num determinado sistema lógico. Desta forma, obtém-se uma *especificação algébrica* do sistema, que consiste na classe de todas as álgebras nesta assinatura que satisfaçam os requisitos pretendidos, isto é, na classe dos possíveis programas para a realização do sistema. A escolha de uma álgebra de entre esta classe para a implementação do sistema, é feita segundo a metodologia do *refinamento passo-a-passo*, que consiste no processo pelo qual se refina esta classe de álgebras de forma gradual a partir da introdução de decisões de implementação, até se encontrar a descrição precisa do programa pretendido. Esta descrição consiste numa nova especificação com uma classe de modelos mais pequena, podendo mesmo, em alguns casos, conter uma única álgebra a menos de isomorfismos. Este refinamento tem que ser feito de forma a que a classe de modelos resultante do processo (ou pelo menos o seu reduto por um morfismo de assinaturas) esteja contida na classe de modelos da especificação original.

Na concepção clássica de especificação algébrica, são usados o conceito de modelação definido no sentido da *relação de satisfação Tarskiana* e a lógica equacional como lógica de suporte ao processo. Contudo, tal como se apresenta de seguida, *software* desenhado segundo o paradigma *da Orientação a Objectos (OO)* requer outras ferramentas mais adequadas para este fim.

A *Orientação a Objectos* é um paradigma da programação que se desenvolveu em resultado de trabalhos nas áreas da Simulação (todos os princípios e conceitos inerentes à *OO*) e da Engenharia de Software (as primeiras implementações neles baseados). Os seus primeiros passos foram dados nos anos 60-70, sendo que o seu impacto ao nível tecnológico se tenha sentido apenas nos anos 90 do século passado. O aumento da complexidade dos sistemas de software despertou desde cedo a necessidade de desenvolver uma metodologia de programação mais organizada e eficiente, tanto ao nível do desenvolvimento, como ao nível da sua análise. Segundo este paradigma, o processo de programação deveria ser efectuado de forma modular, isto é, a partir da composição de unidades de programação mais pequenas denominadas por *módulos*, devendo estas ser encaradas, não pela *forma da sua implementação*, mas *pelas suas funcionalidades-abstracção*. Desta forma, não só se conseguia obter uma visão mais organizada do sistema final obtido, como também se podiam utilizar estes mesmos módulos noutros contextos - *reutilização de componentes de software*. Os primeiros passos neste sentido foram dados pelo aparecimento de linguagens de programação a suportar os conceitos

de *procedimento*, *sub-rotina*, *função* (estruturas compiladas exclusivamente por algum programa principal) e, posteriormente, o de *módulo* (estrutura compilável de forma autónoma). Todas estas componentes podiam, quando necessário, ser evocadas por um qualquer programa, sendo que também se evocavam entre si. Esta primeira abstracção denominada por *abstracção de controlo* continuava ainda muito presa à natureza dos dados a computar, pelo que, por exemplo, seriam necessários dois módulos distintos para efectuar uma soma de inteiros ou de reais. Por outro lado, nesta fase, ainda não havia autonomia real nos módulos, na medida em que o facto de dados definidos num determinado módulo poderem ser utilizados noutros módulos, implicava que, para se utilizar um determinado módulo A que por sua vez evocasse um módulo B, seria necessário utilizar os dois, mais todos os outros de que B dependesse, formando-se, assim, grandes cadeias de interdependência e, por conseguinte, programas pesados e desorganizados. Por forma a que tal não aconteça, ou seja, para que um módulo de software seja verdadeiramente autónomo (e por conseguinte reutilizável em qualquer contexto), o paradigma sugere que todo o módulo deva possuir uma estrutura de dados interna, manipulada exclusivamente à custa das suas funções e às quais só se tenha acesso (utilizador/outros módulos) via um conjunto de operações, denominada no contexto da informática por *interface*. Desta forma, os módulos passam a ser vistos como *mecanismos de abstracção de dados*. Perante o paradigma *OO*, um *objecto* é um módulo de software que possui um *identificador único*, um *estado interno* e um conjunto de operações que podem aceder a esse mesmo estado, podendo desta forma ser visto como uma *cápsula* que esconde do exterior os seus dados internos e a sua implementação - propriedade de *encapsulamento de dados*. Esta técnica é fundamental não só no sentido do desenvolvimento eficiente de software modular, como também na protecção de dados confidenciais e do próprio código de implementação. Observe-se que desta forma, um *objecto OO* pode ser visto como uma *máquina de estados* cujas transições se fazem a partir das operações do módulo. Uma apresentação do paradigma *OO*, com referência a outros conceitos fulcrais tais como *classe*, *hierarquia*, *processo de modelação OO*, etc., pode ser encontrada em [Rum91]. Uma apresentação das origens do paradigma pode ser encontrada, por exemplo, em [Mar00].

A existência de dados encapsulados nos sistemas desenhados segundo este paradigma, sugere uma adaptação dos conceitos de modelação estrita e da satisfação *Tarskiana*,

mais adequados ao processo de especificação algébrica de programas com estas características. Observe-se, por exemplo, que do ponto de vista do utilizador, dois elementos encapsulados podem ser "considerados como iguais" se devolverem os mesmos *outputs* quando submetidos às mesmas computações. Por outro lado, duas implementações podem ser "consideradas como equivalentes" se responderem da mesma forma perante as mesmas computações. Para adequar estes pressupostos à *abordagem algébrica do desenvolvimento de software*, faz-se uma separação entre os géneros da assinatura da especificação: consideram-se os *géneros observáveis*, para representar os dados efectivos a que se tem acesso directo e os *géneros não observáveis*, para representar os dados encapsulados. Neste contexto, uma computação de resultado observável é vista como um termo de resultado observável, ou seja, um termo cujo género esteja entre os ditos géneros observáveis. Por forma a conseguir uma semântica precisa de programas com dados encapsulados, a abordagem observacional/comportamental sugere que se substitua a relação de *igualdade estrita* pela *relação de igualdade observacional*,[1] a partir da qual, dois elementos não observáveis se consideram *observacionalmente iguais*, se tiverem os mesmos comportamentos perante as mesmas computações de resultado visível, isto é, se produzirem os mesmos *outputs*. Perante esta adaptação, para que uma álgebra seja modelo observacional de uma especificação, é suficiente que satisfaça os requisitos comportamentalmente observáveis, podendo não satisfazer estritamente os não observáveis (propriedades internas da especificação). Este facto facilita a prática de reutilização de componentes de *software*, na medida em que torna menos restritiva a concepção de satisfatibilidade standard.

Observe-se que para verificar a igualdade observacional entre dois elementos, é necessário verificar o seu comportamento perante todas as possíveis computações sobre eles executáveis, que geralmente, existem em número infinito. Contudo, em grande parte dos casos, é possível definir um subconjunto dessas possíveis computações, suficiente para efectuar este tipo de prova, existindo na literatura vários métodos para este efeito. É também a partir destes métodos que se prova a correcção observacional de implementações, isto é, que se prova que um determinado programa satisfaz observacionalmente uma dada especificação. Grande parte dos trabalhos apresentados sobre este assunto são no sentido de desenvolver algoritmos eficientes para este tipo de verificação. São vários os grupos de investigação a estudar este tema, existindo neste

[1] Esta relação foi formalizada pela primeira vez por *H. Reichel*;

momento, algumas ferramentas de verificação automática a suportar este tipo de prova (como são os casos do BOBJ, CafeOBJ e SPIKE). Contudo, apesar de em resultado destes trabalhos se terem desenvolvido métodos de verificação observacional com elevados graus de eficiência, prova-se que não existem algoritmos que verifiquem todos os teoremas observacionais em todas as especificações. A incompletude destes métodos deve-se ao facto do *problema da satisfação observacional* se encontrar na *hierarquia aritmética* na classe dos problemas Π_2^0-Hard.

O tema central da presente dissertação é o estudo do processo de especificação algébrica e verificação de programas orientados a objectos via relação de *igualdade observacional*, sendo o objectivo principal do trabalho investigar, cruzar e uniformizar resultados publicados por diferentes fontes sobre este tema, privilegiando, sempre que possível, o rigor e formalismos inerentes a um texto matemático. Grande parte dos trabalhos disponíveis na literatura sobre esta temática são concebidos e apresentados com o intuito de resolver problemas concretos de aplicação prática, tais como, o estudo de técnicas para a verificação e demonstração de propriedades observacionais em especificações de software. Para este feito, os autores trabalham essencialmente com a igualdade observacional total em especificações *flat* (especificadas por um conjunto de equações). São exemplos desta abordagem os trabalhos de *J. Goguen, G. Malcolm, de G. Roşu*, de *A. Bouhoula*, *R. Diaconescu*, *K. Futatsugi*, *P. Padawitz* entre outros.[2] No corrente texto, parte-se de uma abordagem mais universalista, onde os conceitos e teoria subjacentes ao tema são apresentados de um ponto de vista mais abstracto, nomeadamente ao nível da igualdade comportamental parcial (relação de congruência parcial) e das especificações estruturadas. Esta abordagem foi desenvolvida por *M. Bidoit* e *R. Hennicker*.[3] Os conceitos de igualdade observacional total e especificação *flat*, são depois apresentados como casos particulares dos conceitos de igualdade comportamental parcial e de especificação estruturada, respectivamente.

Um tópico bastante estudado neste texto é o do *processo de refinamento passo-a-passo* e, em particular, a sua adaptação à semântica observacional de sistemas. Numa grande parte dos trabalhos publicados sobre a abordagem algébrica ao desenvolvimento de software, impõe-se que a assinatura da especificação inicial do sistema, seja preservada durante o processo de refinamento. Nesta dissertação, estuda-se também o caso

[2] cf. [GM00, GM99, Roş00, BR02, DF00, Pad98].
[3] cf. [BH96, Hen97];

onde estas assinaturas possam variar durante o processo via morfismos de assinaturas e adapta-se este conceito à semântica observacional de sistemas. Um tópico pouco estudado na literatura que aqui se trabalha, é o caso onde se admitem variações no conjunto dos géneros observáveis durante o processo de refinamento, isto é, o caso onde são admitidos encapsulamentos e desencapsulamentos de dados durante este processo. Sendo neste momento, o interesse do estudo deste tópico puramente teórico, discutem-se aqui possíveis aplicações deste estudo na prática, nomeadamente aos níveis da verificação e protecção de código e dados.

Estuda-se também a semântica observacional de sistemas via *relação de* Γ*-igualdade observacional*. Enquanto que na definição de igualdade observacional se consideram todas as possíveis computações (efectuadas via input), na relação de Γ-igualdade observacional são apenas consideradas computações construídas a partir de um subconjunto Γ do conjunto das operações da assinatura da especificação. Apresentam-se neste contexto alguns dos mais importantes algoritmos de verificação observacional desenvolvidos até ao momento, nomeadamente os métodos *Hidden Coinduction*, *Behavioural Coinductive Rewriting*, *Circular Coinductive Rewriting* e *Conditional Circular Coinductive Rewriting with Case Analysis*.

Outro tópico estudado no presente trabalho é o da abordagem da *Lógica Algébrica Abstracta* à semântica observacional de sistemas, apresentada nos trabalhos de *M.Martins* e *D.Pigozzi*. Esta abordagem tem por objectivo investigar a aplicação de conhecimentos desta área da lógica na especificação algébrica de sistemas. Um dos seus resultados fulcrais é o facto da igualdade observacional poder ser vista como um caso particular da *Congruência de Leibniz*, entidade central da *Lógica Algébrica Abstracta*. Apresentam-se neste texto alguns resultados desta abordagem, nomeadamente, uma generalização do método da *Hidden Coinduction* para o caso da *Congruência de Leibniz* e uma caracterização da *especificabilidade observacional de lógicas*, ou seja, uma caracterização do facto de dada uma determinada lógica, saber se é possível definir uma outra, tal que, toda a propriedade observacionalmente válida na primeira, o seja em sentido estrito na segunda. Esta caracterização é feita à custa do estudo dos *Sistemas de Equivalência*, outro importante conceito da *Lógica Algébrica Abstracta*.

Para terminar a dissertação, estuda-se um importante resultado provado por *S. Buss* e *G. Roşu* que situa o *problema da satisfação comportamental* como sendo um problema não recursivamente nem co-recursivamente enumerável, isto é, que estabelece o problema de decidir se uma fórmula é propriedade observacional de uma especificação

como não recursivamente nem co-recursivamente enumerável. Este facto não só implica a inexistência de algoritmos que verifiquem todas as propriedades comportamentais em qualquer especificação, como também a inexistência de algoritmos que o refutem. A demonstração deste facto, consiste na redução do problema da satisfação observacional numa especificação *flat* a um problema Π^0_2-completo. Por forma a expor este assunto, revêm-se de forma sucinta, alguns conceitos relativos à teoria da computabilidade, tais como *Máquinas de Turing*, *Funções Parcialmente Recursivas* e *Hierarquia Aritmética*.

Conclui-se a dissertação com a apresentação de alguns tópicos para desenvolvimento futuro, sugerindo-se neste contexto, uma formalização alternativa do conceito de refinamento influenciada pelos trabalhos da *Lógica Algébrica Abstracta*, nomeadamente, no conceito de *tradução lógica*.

O presente texto está organizado da seguinte forma:

Capítulo 2 - Preliminares. Apresentam-se neste Capítulo alguns conceitos e resultados fundamentais para o acompanhamento da tese, nomeadamente ao nível da *Álgebra Universal (Heterogénea)* (Secção 2.1). Introduzem-se também as noções de *especificação algébrica estruturada* (Secção 2.3), *processo de desenvolvimentos de software por refinamento passo-a-passo* (Secção 2.4), *sistemas de prova em especificações estruturadas* (Secção 2.5) e *sistemas de reescrita* (Secção 2.6).

Capítulo 3 - Igualdade Comportamental. Formaliza-se neste Capítulos o conceito de *igualdade observacional* e adaptam-se alguns conceitos da lógica de primeira ordem a este novo paradigma (Secção 3.1.1). Este trabalho é feito, numa primeira fase, a um nível mais geral, nomeadamente ao nível das *igualdades comportamentais* (relações de congruência parcial)(Secção 3.1).

Define-se um novo operador de especificações algébricas (estruturadas) adequado ao paradigma comportamental (Secção 3.2).

Faz-se uma breve referência à outra importante abordagem à semântica observacional de sistemas - a *abordagem abstractor*. Apresentam-se alguns resultados que estabelecem a ponte entre esta e a abordagem central do texto (Secção 3.3).

Apresentam-se os conceitos de *refinamento comportamental* (Secção 3.4) e de *refinamento observacional* e discutem-se algumas considerações sobre este assunto,

nomeadamente a caracterização do caso onde se admitem variações no conjunto dos géneros observáveis durante o processo de *refinamento passo-a-passo* (Secção 3.4.1).

Apresenta-se um método de verificação de teoremas comportamentais desenvolvido por *Bidoit-Hennicker* - O método do *Operador Lift* (Secção 3.5.1).

Termina-se o Capítulo com uma adaptação do método de prova apresentado na Secção 2.5 para a prova de teoremas comportamentais (Secção 3.6).

Capítulo 4 - O Caso *flat* Observacional - Outras Abordagens. Numa primeira parte do Capítulo apresenta-se a abordagem da *igualdade* Γ-*observacional* de *Goguen-Roşu*. Faz-se aqui uma apresentação retrospectiva dos métodos de verificação observacional propostos por este grupo (Secção 4.1);

Numa segunda parte do Capítulo apresenta-se a abordagem da *Lógica Algébrica Abstracta* à semântica observacional de sistemas (pelos trabalhos de *Pigozzi-Martins*). Nesta parte, apresenta-se a igualdade observacional como um caso particular da *congruência de Leibniz* e generaliza-se o método da *hidden coinduction* para esta relação. Apresenta-se também o conceito de *lógica observacionalmente especificável* e apresenta-se uma caracterização desta propriedade numa dada lógica, via propriedades dos *sistemas de equivalência* (Secção 4.2).

Capítulo 5 - Incompletude das Lógicas Comportamentais. Apresenta-se neste Capítulo um resultado de *Buss* e *Roşu*, que estabelece o problema da satisfação observacional na hierarquia aritmética.

Numa primeira fase da Secção apresentam-se os conceitos fundamentais da teoria da recursividade e computabilidade para o acompanhamento da Secção, nomeadamente os conceitos de *função (parcialmente) recursiva e computável*, *conjuntos recursivamente enumeráveis* e *hierarquia aritmética* (Secção 5.1).

Termina-se o Capítulo com a apresentação de um exemplo de uma especificação onde o problema da satisfação observacional está em Π^0_2-Hard. Esta prova é feita pela redução do problema da satisfação observacional nesta especificação, a um problema Π^0_2-completo (Secção 5.2).

Capítulo 6 - Conclusões e Desenvolvimentos Futuros. Neste Capítulo faz-se um balanço do trabalho realizado e sugerem-se alguns tópicos para desenvolvimentos futuros desta investigação.

Capítulo 2

Preliminares

Introduzem-se neste Capítulo alguns conceitos que estão na base dos assuntos trabalhados no texto, nomeadamente os conceitos subjacentes à álgebra heterogénea, morfismos entre assinaturas, especificações algébricas estruturadas e refinamentos.

2.1 Álgebras Heterogéneas

A procura de uma homogeneização na apresentação de vários conceitos tais como congruência, homomorfismo, produto etc., estudados em diversas estruturas algébricas (grupos, anéis, etc.), está na base do aparecimento de uma das mais estruturantes disciplinas da matemática moderna - a *Álgebra Universal*. Aqui, todos estes conceitos são estudados numa entidade mais abstracta - *a álgebra*- particularizando-se depois os resultados a cada estrutura algébrica. Para tal, entende-se uma álgebra como uma par $\mathbf{A} = (A, \mathcal{F})$, onde A representa o universo da álgebra (ou conjunto suporte), e $\mathcal{F}$ uma família de funções.

Os primeiros registos de trabalhos no sentido desta universalização remotam aos finais do século XIX sendo, contudo, consensual, que a álgebra universal se distingue como ramo independente da Matemática em resultado dos trabalhos de *Birkhoff* nos anos 30 do século passado. Desde então, a sua evolução tem sido constante e estende-se desde os trabalhos de *Alfred Tarski*, ao nível da *Teoria de Modelos* e do *Cálculo Equacional* nas décadas de 50 e 60, até ao estudo das ligações entre Álgebra Universal e a Lógica, investigadas na actualidade pela *Lógica Algébrica Abstracta*. A sua aplicação ao nível da verificação e desenvolvimento de sistemas de software, reflecte-se na prática da *especificação algébrica de sistemas*, na qual programas são modelados

por álgebras (Secção 2.3). Todo o trabalho desenvolvido no presente texto é baseado nesta abordagem.

A Álgebra Heterogénea Universal (*Universal Sorted Algebra*) é uma generalização da teoria da Álgebra Universal mais adequada à especificação algébrica de sistemas. Aqui, em vez de se considerar um único conjunto suporte para a álgebra, é considerada uma família de conjuntos (ou um conjunto heterogéneo), sendo as funções definidas entre estes conjuntos. Este formalismo adequa-se mais às necessidades da programação, onde normalmente se trabalha com dados de vários géneros (por exemplo, naturais, boolenos, strings, etc). Nesta Secção é apresentada a teoria da álgebra universal heterogénea essencial para o acompanhamento do texto.

Uma apresentação da teoria da álgebra universal (caso homogéneo) pode ser encontrada em [BS81] e em [Grä79] (texto mais detalhado). Para uma apresentação da abordagem heterogénea reporta-se o leitor para [STar, LEW00, MT92].

Definições

Seja S um conjunto não vazio cujos elementos são denominados por géneros. Um *S-conjunto* A é uma família de conjuntos indexados por S e denotado por $(A_s)_{s\in S}$. O S-conjunto diz-se vazio se para todo $s \in S$, $A_s = \emptyset$. Um S-conjunto diz-se *localmente finito* se para todo o $s \in S$, A_s for um conjunto finito, e diz-se *globalmente finito* se for localmente finito e tal que $A_s = \emptyset$ excepto num número finito de géneros. Observe-se que no caso de S ser finito, todo o S-conjunto localmente finito é globalmente finito. A generalização dos conceitos da teoria de conjuntos tais como união, intersecção, produto Cartesiano, inclusão e igualdade é feita componente a componente da seguinte forma: para A e B S-conjuntos:

$$A \cup B = (A_s \cup B_s)_{s\in S}$$

$$A \cap B = (A_s \cap B_s)_{s\in S}$$

$$A \times B = (A_s \times B_s)_{s\in S}$$

$$A \subseteq B \text{ sse } A_s \subseteq B_s \text{ para todo o } s \in S$$

$$A = B \text{ sse para todo } s \in SA_s = B_s.$$

Definição 2.1.1 (Função heterogénea)**.** *Sejam $A = (A_s)_{s\in S}$ e $B = (B_s)_{s\in S}$ conjuntos heterogéneos. Uma S-*função heterogénea *$f : A \to B$, consiste numa S-família de funções $(f_s)_{s\in S}$ definida para cada $s \in S$ como uma função $f_s : A_s \to B_s$.*

Definição 2.1.2 (Relação binária heterogénea)**.** *Seja $A = (A_s)_{s\in S}$ um S-conjunto. Uma S-*relação binária *$R \subseteq A\times A$ consiste numa S-família de relações binárias $(R_s)_{s\in S}$ tal que $R_s \subseteq A_s \times A_s$. Normalmente, escreve-se aR_sa' em vez de $(a, a') \in R_s$.*

Definição 2.1.3 (Relação de equivalência heterogénea)**.** *Sejam A um S-conjunto e R uma S-relação binária. A relação R é uma* relação de equivalência *em A se para todo o $s \in S$:*

- *R_s é reflexiva, i.e, aR_sa, para todo o $a \in A_s$;*
- *R_s é simétrica, i.e., se aR_sa' então $a'R_sa$;*
- *R_s é transitiva, i.e., se aR_sa' e $a'R_sa''$, então aR_sa''.*

Para um elemento $a \in A_s$, define-se classe de equivalência de a modulo R *como o conjunto $a/R_s = \{b \in A_s|aR_sb\}$. O* conjunto quociente de A por R *é o S-conjunto $A/R = (A_s/R_s)_{s\in S}$ tal que $A_s/R_s = \{a/R_s|a \in A_s\}$.*

Definição 2.1.4 (Assinatura heterogénea)**.** *Uma* assinatura heterogénea Σ*, é um par (S, Ω), onde:*

- *S é um conjunto (de nome de géneros);*
- *Ω é um $(S^* \times S)$-conjunto (de nomes de operação);*

onde S^* representa o conjunto de sequências finitas de elementos de S.

Os símbolos de função de $\Omega_{s_1\cdots s_n,s}$ dizem-se símbolos de função com *argumentos de géneros* $s_1 \cdots s_n$ e *resultado de género* s. Os elementos de $\Omega_{\epsilon,s}$ dizem-se *constantes de género* s (onde ϵ representa a sequência vazia de S^*). Um símbolo de função $f \in \Omega_{s_1\cdots s_n,s}$ pode também ser representado por $f : s_1, \ldots, s_n \to s \in \Sigma$, ou simplesmente $f : s_1, \ldots, s_n \to s$. Identifica-se no seguimento do texto o conjunto Ω pelo conjunto $\bigcup_{\omega\in S^*,s\in S} \Omega_{\omega,s}$. Uma assinatura $\Sigma' = (S', \Omega')$ diz-se *subassinatura* de uma assinatura $\Sigma = (S, \Omega)$ se $S' \subseteq S$ e $\Omega'_{\omega,s} \subseteq \Omega_{\omega,s}$ para todos os $\omega \in S^*$ e $s \in S$. Apresentam-se de seguida alguns exemplos de assinaturas heterogéneas. As assinaturas vêm descritas por extenso acompanhadas de diagrama ilustrativos:

Exemplo 2.1.5 (Espaços vectoriais)**.** *O conceito de espaço vectorial é conhecido como sendo a generalização natural das noções de espaço bi e tridimensional, nos quais se adicionam vectores e se multiplicam vectores por números reais (escalares). Para*

especificar estas estruturas pode-se recorrer a uma assinatura com dois géneros (o `elt` *relativo aos vectores e o* `esc` *aos escalares), com as seguintes operações:*

```
[GEN]
     elt;
     esc;
[OP]
```

$+_{esc}$`:esc,esc -> esc;`

$*_{esc}$`:esc,esc -> esc;`

1_{esc}`:-> esc;`

0_{esc}`: -> esc;`

$\_^{-1_{esc}}$`:esc -> esc;`

$-_{esc}$`:esc -> esc;`

$+_{elt}$`:elt,elt -> elt;`

$-_{elt}$`:elt -> elt;`

0_{elt}`: -> elt;`

$\times$`:esc,elt -> elt;`

◇

Exemplo 2.1.6 (As assinaturas Σ_{AUT-I} e $\Sigma_{AUT-I/O}$)**.** *Um autómato*[1] *é um máquina abstracta constituída por um* conjunto finito de estados `Z`*, por um conjunto de símbolos de Input* `X`*, por uma constante* $\mathtt{z}_0 \in$ `Z` *e por uma* função de transição δ*:*

```
[GEN]
    Z;
    X;
[OP]
```

δ : `Z,X -> Z` *(referente à função de transição do autómato);*

[1] *O recurso a exemplos relacionados com o conceito de autómato é frequente em todo o texto. Esta escolha deve-se essencialmente ao facto de conceitos bem conhecidos, como é o caso da* equivalência de estados num autómato, equivalência de autómatos *e* autómato reduzido, *aparecerem como casos particulares de conceitos aqui trabalhados (ver Secção 3);*

z_0:-> Z*(referente ao estado inicial do autómato);*

Diz-se que um autómato H *transita do estado* z *para o estado* z' *pelo input* x *quando* δ(z,x)=z', *e define-se* acção do autómato $\widehat{\delta}$: Z,X^* → Z *como a extensão natural de* δ *a sequências de símbolos de* X *da seguinte forma:*

1. $\widehat{\delta}$(z,ϵ)=z, *para todo o estado* z, *onde* ϵ *representa a palavra vazia;*
2. $\widehat{\delta}$(z,xω)=δ($\widehat{\delta}$(z,ω),x), *onde* z, x *e* ω *representam estado, letra e palavra arbitrários respectivamente,*

Um autómato pode ser utilizado para o reconhecimento de sequências de símbolos de X. *No contexto da teoria da computação, estas sequências são conhecidas por palavras e o conjunto* X *por alfabeto. Uma palavra* $\omega \in X^*$ *é reconhecida por um autómato* H, *se a acção* $\widehat{\delta}(z_0, \omega) \in$ Z', *para um determinado* Z' $\subset$ Z *denominado por* subconjunto de estados finais. *Define-se linguagem reconhecida por um autómato* H *como o conjunto* $L(H) = \{\omega \in X^* | \widehat{\delta}(z_0, \omega) \in Z'\}$.

Uma importante variante destas máquinas são os denominados autómatos com output, *também conhecidos por* máquinas sequenciais. *Um autómato com output consiste num autómato com um* alfabeto de símbolos de output B, *juntamente com uma função* λ : Z, X → B *denominada de* função output. *Estende-se a aplicação de* λ *a sequências de* X^* *pela função* $\widehat{\lambda}$: Z, X^* → B^* *definida por:*

1. $\widehat{\lambda}$(z, ϵ)=ϵ, *onde* ϵ *representa a palavra vazia de* B^*;
2. $\widehat{\lambda}$(z,xω)=λ(z,x)$\widehat{\lambda}$(δ(z,x), ω), *para* $\omega \in X^*$;

A assinatura $\Sigma_{AUT-I/O}$ *dos autómatos com output consiste no seguinte:*

```
[GEN]
    Z;
    X;
    B;
[OP]
```

z_0:->Z*(referente ao estado inicial do autómato);*

δ : Z,X -> Z*(referente à função de transição do autómato);*

λ:Z,X ->B;

Note-se que um autómato de Σ_{AUT-I}, pode ser visto como o caso $\Sigma_{AUT-I/O}$, onde `B` $= \{0, 1\}$ *(rejeita, aceita), e* λ *tal que se* `z` $\in$ `Z'`, λ`(z)`$=1$, *onde* `Z'` *representa o subconjunto de estados finais, razão pela qual, se escolhe trabalhar esta estrutura mais geral, noutros exemplo do decorrer do texto.*

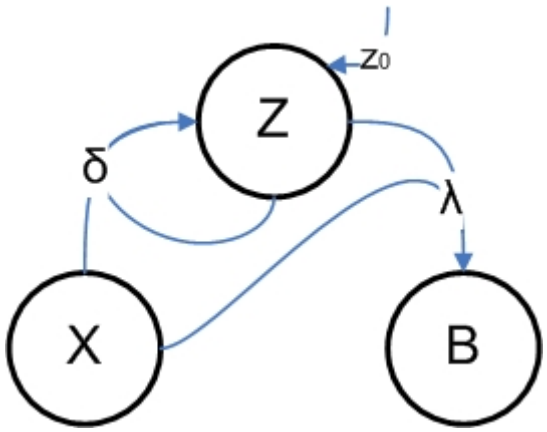

Figura 2.1: Diagrama da assinatura $\Sigma_{AUT-I/O}$.

◇

Exemplo 2.1.7 (A assinatura Σ_{CELL} [Roş00])**.** *Considere-se um sistema do tipo de uma célula de memória de um computador, na qual se podem armazenar e ler valores. Este caso pode ser especificado a partir da assinatura $\Sigma = (S, \Omega)$ com $S = \{$*`elt`$, $`cell`$\}$*, onde* `elt` *representa o género dos valores a armazenar na célula e* `cell` *o das células propriamente ditas e, $\Omega = \{$*`put`$, $`get`$\}$*, onde* `put` *denota a função que armazena um valor* `elt` *numa célula* `cell` *e* `get` *a função que lê o valor armazenado em* `cell`*.*

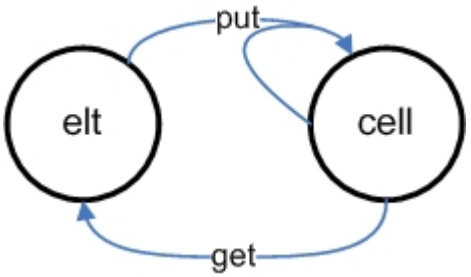

Figura 2.2: Diagrama da assinatura Σ_{CELL}.

`[GEN]`

```
    elt;
    cell;
[OP]
    put: elt,cell -> cell;
    get:cell -> elt;
```

◇

Exemplo 2.1.8 (A assinatura Σ_{STATE} [HWB97]). *Considere-se agora um sistema mais complexo, onde se tem vários identificadores (conjunto* `ID`*), que podem identificar elementos (género* `elt`*). Fazendo uma analogia com o exemplo anterior, pode-se agora pensar num disco duro, onde cada célula de memória é identificada por um elemento de um conjunto de moradas* `id`*. A função* `update`*, escreve um valor de* `elt` *na célula com uma morada de* `id`*, e a função* `lookup`*, vai ler o valor* `elt` *da célula com morada de* `id`*. A constante* `init` *do género* `state` *pode ser necessária para identificar um estado inicial.*

```
[GEN]
    state;
    id;
    elt;
[OP]
    init: -> state;
    update:id,elt,state -> state;
    lookup: id,state -> elt;
```

◇

Segue a definição de um importante conceito da *Álgebra Universal-a Assinatura das operações derivadas.* Intuitivamente, tem-se que dada uma assinatura heterogénea $\Sigma = (S, \Omega)$, uma *operação derivada* de Σ é um termo de $T_\Sigma(\{z_1, \ldots, z_n\})$, para $z_1, \ldots, z_n$ variáveis especiais dos géneros $s_1, \ldots, s_n \in S$. A *Assinatura das operações derivadas de* Σ é a assinatura composta por estas operações nos géneros de Σ:

Definição 2.1.9 (Assinatura das operações derivadas). *Seja* $\Sigma = (S, \Omega)$ *uma assinatura heterogénea. Para qualquer sequência* $s_1 \cdots s_n \in S^*$*, seja* $I_{s_1 \cdots s_n}$ *uma* S*-família de conjuntos* $\bar{1} : s_1, \ldots, \bar{n} : s_n$*. Define-se* assinatura das operações derivadas de Σ*,*

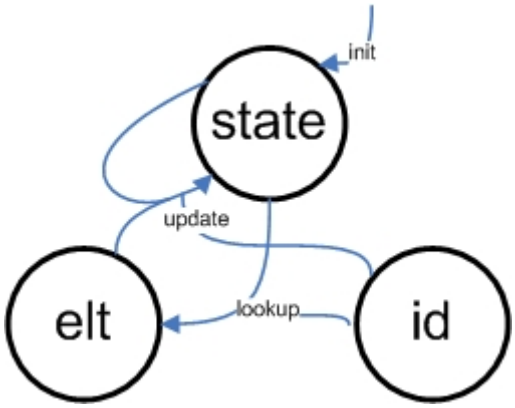

Figura 2.3: Diagrama da assinatura Σ_{STATE}.

como sendo a assinatura $Der(\Sigma) = (S, \Omega^{der})$*, onde para todo* $s_1 \cdots s_n \in S^*$*,* $s \in S$*,* $\Omega^{der}_{s_1 \cdots s:n,s} = T_\Sigma(I_{s_1 \cdots s_n})_s$.

Para algumas considerações sobre este assunto consultar [STar, Secção1.5.2.].

Σ-álgebras

As Σ-álgebras desempenham o papel de interpretação semântica das assinaturas:

Definição 2.1.10 (Σ-álgebra heterogénea)**.** *Para a assinatura* $\Sigma = (S, \Omega)$*, uma* Σ*-álgebra* A *consiste:*

- *num* S*-conjunto* $A = (A_s)_{s \in S}$*, onde para todo* $s \in S$*,* A_s *denota um conjunto não vazio denominado de* domínio de género s*;*
- *para cada símbolo* $f : s_1, \ldots, s_n \to s \in \Sigma$*, uma função* $f^A : A_{s_1} \times \cdots \times A_{s_n} \to A_s$.

O conjunto de todas as Σ-álgebras denota-se por $Alg(\Sigma)$.

De seguida apresentam-se alguns exemplos de Σ-álgebras relativas às assinaturas Σ apresentadas:

Exemplo 2.1.11 (Σ_{VECTOR}-álgebras)**.** *São exemplos de* Σ_{VECTOR}*-álgebras os conhecidos espaços vectoriais reais.*

◇

Exemplo 2.1.12 (Σ_{AUT-I}-álgebra)**.** *Considere-se a* Σ_{AUT-I}*-álgebra* A *tal que* $A_X = \{0, 1\}$*,* $A_Z = \{e_0, e_1, e_2, e_3\}$*,* $\mathsf{z}^A_0 = e_0$*,* e

δ^A	*0*	*1*
e_0	e_3	e_1
e_1	e_3	e_2
e_2	e_2	e_2
e_3	e_3	e_3

Para $\mathtt{Z'} = \{e_2\}$*, a* Σ_{AUT-In}*-álgebra A representa o autómato que reconhece exactamente as palavras de* $(A_{\mathtt{X}})^*$ *começadas por* 11*.*

Observe-se que um autómato é assim definido pelo par $\langle H, Z' \rangle$*, onde H é uma* Σ_{aut-I}*-álgebra e* $\mathtt{Z'} \subseteq \mathtt{Z}^A$*, na medida em que, por forma a que qualquer autómato seja uma* Σ_{aut-I}*-álgebra, não se podem distinguir como constantes os estados finais de H. Esta limitação pode ser contornada pela introdução na assinatura de um predicado* booleano *para a distinção dos estados finais. Uma outra forma natural de tratar estes casos é pelo uso de* Order Sorted Algebras (*cf.* [GD94a]) *ou, no contexto da* lógica algébrica abstracta*, por matrizes* (*cf. [BP92]*)*.*

◇

Exemplo 2.1.13 (Σ_{CELL}-álgebras ([Roş00]))**.** *Seguem-se dois exemplos de* Σ_{CELL}*-álgebras. Considere-se o S-conjunto* $A_{\mathtt{elt}} = \mathbb{N}$*,* $A_{\mathtt{cell}} = \mathbb{N}$*, com as seguintes funções:*

$$\mathtt{get}^A(n) = n;$$

$$\mathtt{put}^A(m, n) = m.$$

Considere-se agora o S-conjunto B tal que $B_{\mathtt{elt}} = \mathbb{N}$*,* $B_{\mathtt{cell}} = \mathbb{N}^*$*, com as seguintes funções:*

$$\mathtt{get}^B(\epsilon) = 0;$$

$$\mathtt{get}^B(n\omega) = n;$$

$$\mathtt{put}^B(m, \omega) = m\omega,$$

onde $\epsilon, \omega \in \mathbb{N}^*$ *representam a palavra vazia e uma lista de naturais respectivamente. Esta segunda álgebra pode ser importante, por exemplo, no caso em que se pretende guardar o histórico dos valores inscritos numa célula.*

◇

De seguida, apresentam-se alguns conceitos da álgebra universal, adaptados ao caso heterogéneo, todos eles utilizados no processo de especificação algébrica de sistemas abordado no trabalho:

Definição 2.1.14 (Homomorfismo)**.** *Sejam $\Sigma = (S, \Omega)$ uma assinatura e A e B Σ-álgebras. Um* Σ-homomorfismo *$h : A \to B$ é uma S-família de funções $h_s : A_s \to B_s$ tal que, para toda o símbolo de função $f : s_1, \ldots, s_n \to s \in \Sigma$ e para todos os $a_1 \in A_{s_1}, \ldots, a_n \in A_{s_n}$, se tem que $h_s(f^A(a_1, \ldots, a_n)) = f^B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$.*

Um homomorfismo injectivo denomina-se por *monomorfismo* e um homomorfismo injectivo por *epimorfismo*. Um homomorfismo é um *isomorfismo* se for monomorfismo e epimorfismo. Duas álgebras A e B dizem-se *isomorfas*, em símbolos $A \cong B$, se existir um isomorfismo entre elas.

Definição 2.1.15 (Relação de congruência parcial)**.** *Sejam $\Sigma = (S, \Omega)$ uma assinatura e A uma Σ-álgebra. Uma* Σ-congruência parcial em *A, é uma S-família de relações simétricas e transitivas (relações de equivalência parcial) não vazias $\approx^A = (\approx^A_s)_{s \in S}$, tais que para qualquer $f : s_1, \ldots, s_n \to s \in \Sigma$, para quaisquer $a_i, b_i \in A_{s_i}$, $1 \leq i \leq n$ se $a_i \approx^A_{s_i} b_i$, então $f^A(a_1, \ldots, a_n) \approx^A_s f^A(b_1, \ldots, b_n)$, isto é, f é* compatível *com $\approx^A$ (ver figura 2.1).*

Quando para todo o $s \in S$, as relações $\approx^A_s$ são também reflexivas, isto é, quando para todo $a \in A_s$,$a \approx^A_s a$, a relação $\approx^A$ diz-se relação de Σ-congruência total*, ou simplesmente* relação de Σ-congruência*. O* domínio de definição *da congruência parcial $\approx^A$ é denotado por $Dom(\approx^A)$, e define-se como $Dom(\approx^A)_s = \{a \in A_s | a \approx^A_s a\}$ para todo $s \in S$.*

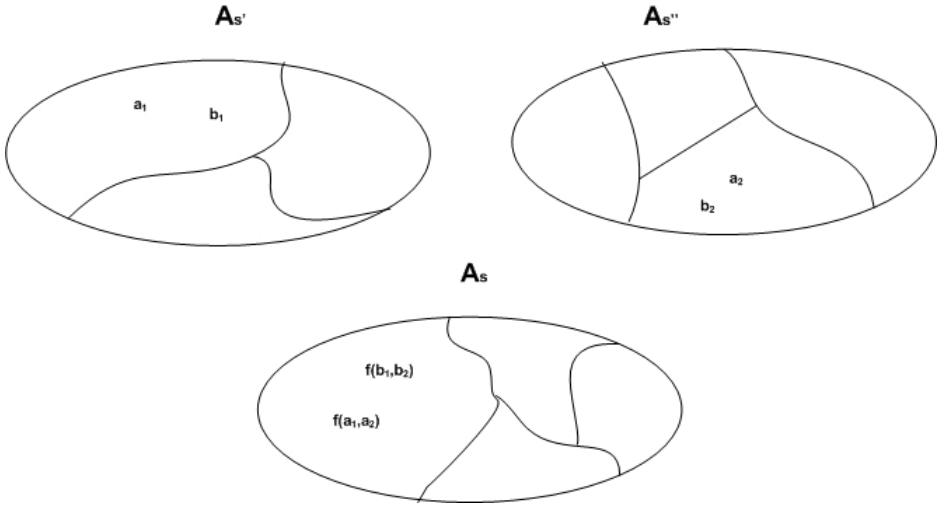

Figura 2.4: Compatibilidade de uma função com uma relação de congruência.

Definição 2.1.16 (Σ-álgebra quociente)**.** *Sejam A uma Σ-álgebra e $\approx$ uma Σ-congruência em A. A* álgebra quociente de A *por $\approx$ é a Σ-álgebra $A/\approx$ definida da seguinte forma:*

- $(A/\approx)_s = A_s/\approx^A_s$, *para todo o* $s \in S$;
- *para toda* $f : s_1, \ldots, s_n \rightarrow s \in \Sigma$, *e para todos os* $a_1/\approx^A_{s_1} \in A_{s_1}/\approx^A_{s_1}, \ldots, a_n/\approx^A_{s_n} \in A_{s_n}/\approx^A_{s_n}$, $f^{A/\approx}(a_1/\approx^A_{s_1}, \ldots, a_n/\approx^A_{s_n}) = f(a_1, \ldots, a_n)/\approx^A_s$.

No caso de $\approx$ ser relação de congruência parcial, define-se álgebra quociente de A por $\approx$ *como o quociente de $Dom(\approx^A)$ por $\approx^A$, e denota-se também por $A/\approx$.*

Formaliza-se de seguida os conceitos de *variável* e *conjunto de variáveis* para uma assinatura. Para tal, associa-se a cada assinatura heterogénea $\Sigma = (S, \Omega)$ uma S-família $V = V_{s \in S}$ de conjuntos de tamanho infinito disjuntos dois a dois. Um elemento de V_s denomina-se por *variável de género* s, e a uma S-família $X \subseteq V$ por *conjunto de variáveis para a assinatura* Σ. Convenciona-se aqui que os elementos de V tenham nomes diferentes dos nomes dos elementos de Ω.

Definição 2.1.17 (Conjunto $T_\Sigma(X)$)**.** *Sejam $\Sigma = (S, \Omega)$ uma assinatura heterogénea e X um conjunto de variáveis para Σ. Para cada $s \in S$, define-se* $(T_\Sigma(X))_s$, conjunto de termos do género s, *como o menor conjunto $T_\Sigma(X)$tal que:*

- *Para todo o $s \in S$ e $x \in X_s$ tem-se que $x \in (T_\Sigma(X))_s$;*
- *Se existe uma função $f :\rightarrow s \in \Sigma$, então $f \in (T_\Sigma(X))_s$ (constantes);*
- *Para todo o $f : s_1, \ldots, s_n, \rightarrow s \in \Sigma$, e para todo termo $t_i \in T_\Sigma(X)_{s_i}, 1 \leq i \leq n$, tem-se que $f(t_1, \ldots, t_n) \in T_\Sigma(X)_s$.*

$Var(t)$ denota o conjunto de variáveis que ocorrem no termo t.

É bem sabido que sempre que $T_\Sigma(X) \neq \emptyset$ é possível definir de forma natural operações sobre este conjunto por forma a obter uma Σ-álgebra com o conjunto de suporte $T_\Sigma(X)$ [STar]. Esta Σ-álgebra é denominada por *álgebra dos termos*.

O conjunto de fórmulas que se define a seguir, é uma adaptação do conjunto de fórmulas de primeira ordem ao caso heterogéneo:

Definição 2.1.18 (Fórmulas)**.** *Sejam $\Sigma = (S, \Omega)$ uma assinatura heterogénea e X um conjunto de variáveis para Σ. O conjunto das Σ-fórmulas denota-se por $Flas(\Sigma)$ e é o menor conjunto tal que:*

1. *Se* $t, r \in T_\Sigma(X)_s$, *então* $t{=}r \in Flas(\Sigma)$;

2. *Se* $\psi, \phi \in Flas(\Sigma)$ *então* $\neg\phi$ *e* $\psi \wedge \phi \in Flas(\Sigma)$;

3. *Se* $\psi \in Flas(\Sigma)$ *e* x *é uma variável de tipo* s *então* $\forall x{:}s.\psi \in Flas(\Sigma)$;

4. *Se* $\{\psi_i | i \in I\}$ *é uma família contável de* Σ*-fórmulas, então* $\bigwedge_{i\in I} \psi_i \in Flas(\Sigma)$ *e* $\bigvee_{i\in I} \psi_i \in Flas(\Sigma)$. *Caso* $\{\psi_i | i \in I\}$ *infinito, as fórmulas* $\bigwedge_{i\in I} \psi_i$ *e* $\bigvee_{i\in I} \psi_i$ *dizem-se infinitárias.*

Uma Σ*-fórmula do tipo 1. é denominada por* equação, *e o conjunto de todas as* Σ*-equações denota-se por* $Eq_\Sigma(X)$;

Observe-se que os conectivos $\vee$, $\rightarrow$ e $\exists$ são definidos de forma usual à custa dos casos em cima mencionados.

Definição 2.1.19 (Funções de valoração e de interpretação)**.** *Sejam* $\Sigma = (S, \Omega)$ *uma assinatura heterogénea,* X *um conjunto de variáveis para* Σ *e* A *uma* Σ*-álgebra. Uma* função de valoração $\alpha : X \rightarrow A$ *é uma* S*-família de funções* $(\alpha_s : X_s \rightarrow A_s)_{s\in S}$. *Uma valoração* α *é unicamente estendida a um homomorfismo* $I_\alpha : T_\Sigma(X) \rightarrow A$ *da seguinte forma:*

1. $I_{\alpha_s}(x) =_{def} \alpha_s(x)$;

2. *Para toda* $f : s_1, \ldots, s_n \rightarrow s \in \Sigma$, *para todo o* $t_i \in T_\Sigma(X)_{s_i}$,

$$I_{\alpha_s}(f(t_1, \ldots, t_n)) =_{def} f^A(I_{\alpha_{s_1}}(t_1), \ldots, I_{\alpha_{s_n}}(t_n)).$$

A função I_α *é denominada por* interpretação induzida por α.

As valorações $\alpha : X \rightarrow T_\Sigma(X)$ denominam-se de *substituições.*

Definição 2.1.20 (Relação de satisfação *Tarskiana* (standard))**.** *Sejam* Σ *uma assinatura heterogénea e* X *um conjunto de variáveis para* Σ. *Define-se a relação* $\models$, *relação de satisfação (standard), da seguinte forma:*

Sejam A *uma* Σ*-álgebra,* $l, r \in T_\Sigma(X)_s$ *dois termos do género* s, ϕ *e* ψ *duas* Σ*-fórmulas,* $\{\phi_i | i \in I\}$ *um conjunto contável de* Σ*-fórmulas e* $\alpha : X \rightarrow A$ *uma função de valoração.*

1. $A, \alpha \models l{=}r$, *se* $I_\alpha(l) = I_\alpha(r)$;

2. $A, \alpha \models \neg\phi$ *se* $A, \alpha \models \phi$ *não se verifica;*

3. $A, \alpha \models \phi \wedge \psi$ *se* $A, \alpha \models \phi$ *e* $A, \alpha \models \psi$;

4. $A, \alpha \models (\forall x : s).\phi$ *se para qualquer função de valoração* $\beta : X \to A$ *tal que* $\beta(y) = \alpha(y)$, *para todo* $y \neq x$ *se tem que* $A, \beta \models \phi$.

A definição da relação de Satisfação *Tarskiana* para fórmulas possivelmente infinitárias*, é obtida pela introdução de:*

- $A, \alpha \models \bigwedge_{i \in I} \phi_i$ *sse para todo o* $i \in I$ *se tem que* $A, \alpha \models \phi_i$;

Escreve-se $A \models \phi$ quando $A, \alpha \models \phi$ para toda a valoração $\alpha : X \to A$, e dada uma classe de Σ-álgebras C, $C \models \phi$ significa que para toda álgebra $A \in C$, $A \models \phi$. Dado um conjunto de Σ-fórmulas Φ escreve-se $A \models \Phi$ se $A \models \phi$ para todo o $\phi \in \Phi$ e, de forma análoga, define-se $C \models \Phi$.

Apresentou-se na Secção anterior o conceito de assinatura das operações derivadas (Definição 2.1.9). Segue-se a definição de *Álgebra das Operações Derivadas*:

Definição 2.1.21 (Álgebra das operações derivadas)**.** *Sejam* $\Sigma = (S, \Omega)$ *uma assinatura e* A *uma* Σ*-álgebra.* A álgebra das operações derivadas *de* A *é uma* $Der(\Sigma)$*-álgebra* A^{der} *definida da seguinte forma:*

- *para todo o* $s \in S$, $A^{der}_s = A_s$;

- *para toda a operação* $f \in \Omega^{der}_{s_1 \cdots s_n, s}$, *e para todo o* $a_1 \in A^{der}_{s_1}, \ldots, a_n \in A^{der}_{s_n}$, $f^{A^{der}}(a_1, \ldots, a_n) = I_\alpha(f)$, *para* $\alpha{:}I_{s_1 \cdots s_n} \to A^{der}$ *tal que* $\alpha(\bar{1}) = a_1, \ldots, \alpha(\bar{n}) = a_n$.

2.2 Morfismos de assinaturas

O conceito de morfismo de assinatura tem um papel central no trabalho de especificação algébrica, nomeadamente ao nível do estudo de refinamentos (Secções 2.4 e 3.4). Introduzem-se nesta Secção alguns conceitos e considerações sobre o assunto:

Definição 2.2.1 (Morfismo de assinaturas)**.** *Sejam* $\Sigma = (S, \Omega)$ *e* $\Sigma' = (S', \Omega')$ *duas assinaturas. Um* morfismo de assinaturas $\sigma : \Sigma \to \Sigma'$, *é um par* $\sigma = (\sigma_{gen}, \sigma_{op})$, *onde* $\sigma_{gen} : S \to S'$ *e* $\sigma_{op} : \Omega \to \Omega'$, *tal que, para cada símbolo de operação* $f : s_1, \ldots, s_n \to s \in \Sigma$, $\sigma_{op}(f)$ *é um símbolo de operação* $\sigma_{op}(f) : \sigma_{gen}(s_1), \ldots, \sigma_{gen}(s_n) \to \sigma_{gen}(s) \in \Sigma'$.

Definição 2.2.2 (Álgebra reduto)**.** *Sejam A' uma Σ'-álgebra e $\sigma : \Sigma \to \Sigma'$ um morfismo de assinaturas. O* σ-reduto *de A' é a Σ-álgebra $A' \restriction_\sigma$ definida da seguinte forma:*

- *Para todo $s \in S$, $(A' \restriction_\sigma)_s = A'_{\sigma(s)}$;*
- *Para toda $f : s_1, \ldots, s_n \to s \in \Sigma$,*

$$f^{A' \restriction_\sigma} = \sigma_{op}(f)^{A'}, \text{ isto é,}$$

$$\text{sendo } f^{A' \restriction_\sigma} : A' \restriction_{\sigma s_1} \times \cdots \times A' \restriction_{\sigma s_n} \to A' \restriction_{\sigma s},$$

$$\text{tem-se } \sigma_{op}(f)^{A'} : A'_{\sigma_{gen}(s_1)} \times \cdots \times A'_{\sigma_{gen}(s_n)} \to A'_{\sigma_{gen}(s)}.$$

Observe-se que se σ for a função identidade, $A \restriction_\sigma$ é a própria álgebra A. Quando se tem $\Sigma \subseteq \Sigma'$, $A' \restriction_\Sigma$ denota a álgebra reduto de A pelo morfismo de inclusão. Neste caso, "são esquecidos"todos os géneros e operações de Σ' que não estão em Σ. Este morfismo é denotado por $\iota : \Sigma \hookrightarrow \Sigma'$. A construção de uma álgebra reduto é feita no Exemplo 2.4.5.

Tal como foi apresentado na Secção 2.1, a cada assinatura $\Sigma = (S, \Omega)$ está associada uma S-família V de conjuntos infinitos disjuntos dois a dois. Assuma-se, sem perda de generalidade que, dado um morfismo de assinaturas $\sigma : \Sigma \to \Sigma'$, a S'-família V' associada a Σ' é tal que, para todo o $s \in S$, $V_s \subseteq V'_{\sigma(s)}$. Assim, a partir do conjunto $X = (X_s)_{s \in S}$ de variáveis para Σ, constrói-se um conjunto de variáveis X' para Σ' como o S'-conjunto definido para cada $s' \in S'$ como $X'_{s'} = \bigcup_{\sigma(s)=s'} X_s$ (cf. [STar, Mar06]). Agora, dada uma assinatura Σ com um conjunto de variáveis X, um morfismo de assinaturas $\sigma : \Sigma \to \Sigma'$ e um termo $t \in T_\Sigma(X)_s$, define-se recursivamente $\sigma(t) \in T'_\Sigma(X')_{\sigma(s)}$ da seguinte forma:

- para todo o $x : s \in X_s$, $\sigma(x : s) = x : \sigma(s)$;
- para toda a constante $f :\to s \in \Sigma$, $\sigma(f) :\to \sigma(s) \in \Sigma'$;
- para toda a operação $f : s_1, \ldots, s_n \to s \in \Sigma$, para todos os $t_i \in T_\Sigma(X)_{s_i}$,

$$\sigma(f(t_1, \ldots, t_n)) = \sigma(f)(\sigma(t_1), \ldots, \sigma(t_n)).$$

Dada uma Σ-equação $\forall X.t \approx t'$, define-se $\sigma(\forall X.t \approx t')$ como sendo a Σ'-equação $\forall X'.\sigma(t) \approx \sigma(t')$ e estende-se σ às fórmulas de $Flas(\Sigma)$ de forma natural (por exemplo $\sigma(\neg\phi) = \neg\sigma(\phi)$, $\sigma(\phi \wedge \psi) = \sigma(\phi) \wedge \sigma(\psi)$, etc).

Para toda a valoração $\alpha' : X' \to A'$, define-se *valoração reduto* de α' como sendo a valoração $(\alpha' \restriction_\sigma) : X \to A' \restriction_\sigma$, tal que $(\alpha' \restriction_\sigma)_s(x : s) = \alpha'_{\sigma(s)}(x : \sigma(s))$.

O lema que se segue, estabelece uma correspondência entre uma valoração e o seu reduto. Este resultado aparece na literatura pelo nome de *Teorema do reduto* (cf. [LEW00, Teorema 5.9]):

Lema 2.2.3. *Sejam Σ e Σ' assinaturas, $\sigma : \Sigma \to \Sigma'$ um morfismo de assinaturas, X um conjunto de variáveis para Σ, X' um conjunto de variáveis para Σ' construído como em cima e t um termo de $T_\Sigma(X)$. Para as valorações $\beta : X' \to A'$ e $\alpha : X \to A' \restriction_\sigma$ tais que $\beta \restriction_\sigma = \alpha$, tem-se que $\alpha(t) = \beta(\sigma(t))$.*

Termina-se a Secção com a apresentação de um resultado central no estudo da especificação algébrica de sistemas:

Lema 2.2.4 (Lema da satisfação)**.** *Se $\sigma : \Sigma \to \Sigma'$ um morfismo de assinaturas,e ϕ uma Σ-equação e A' uma Σ'-álgebra. Então*

$$A' \models \sigma(\phi) \text{ sse } A' \restriction_\sigma \models \phi.$$

A demonstração do lema pode ser encontrada em [GB92, Lema 60]. O seu impacto ao nível da prática da especificação algébrica de sistemas é discutido aquando o estudo do conceito de refinamento nas Secções 2.4 e 3.4.

2.3 Especificações Algébricas Estruturadas

Tal como se referiu na Secção 1, existem duas grandes abordagens ao processo de especificação formal de software: a *orientada a modelos* e a *algébrica* (ou *orientada às propriedades*). Apresenta-se nesta Secção o segundo caso, no qual, programas são vistos como álgebras e os seus termos como computações. Uma *especificação algébrica de software SP* é definida por uma assinatura $Sig(SP)$, e por uma classe de modelos de SP, denotada por $[\![SP]\!]$ e, que seja tal que $[\![SP]\!] \subseteq Alg(Sig(SP))$. As $Sig(SP)$-álgebras de $[\![SP]\!]$ são conhecidas por *realizações correctas* ou *modelos* de SP pois modelam os possíveis programas para a implementação de SP. Formalmente:

Definição 2.3.1 (Especificação algébrica)**.** *Uma* especificação algébrica *SP é um par $(Sig(SP), [\![SP]\!])$, onde $Sig(SP)$ é uma assinatura e $[\![SP]\!]$ uma classe de $Sig(SP)$-álgebras. A classe $[\![SP]\!]$ denomina-se por* classe de modelos de *SP e $Sig(SP)$ por* assinatura de *SP.*

Quando uma fórmula ϕ é satisfeita por todos os modelos de SP, isto é, quando $[\![SP]\!] \models \phi$ diz-se que SP satisfaz ϕ e escreve-se $SP \models \phi$. Duas especificações SP e SP' dizem-se semanticamente equivalentes se $Sig(SP) = Sig(SP')$ e $[\![SP]\!]=[\![SP']\!]$. Uma especificação SP diz-se inconsistente quando não é realizável, isto é, quando $[\![SP]\!] = \emptyset$.

Neste contexto, quando se pretende especificar um determinado sistema de software, define-se uma assinatura adequada aos nomes de géneros e funções apropriados ao sistema pretendido e expressam-se os comportamentos desejados destas operações num determinado sistema lógico. São muitas as especificações construídas desta forma:

Exemplo 2.3.2 (GRUPO). *A expressão que se segue, especifica a classe dos grupos:*

```
Spec GRUPO  =
[GEN]
     elt;
[OP]

0+: -> elt;
(-_):elt -> elt;
+:elt,elt -> elt;

[AX]

(∀a,b,c:elt).(a+b)+c=a+(b+c);
(∀a:elt).a+0+=0++a=a;
(∀a:elt).a+(-a)=0+;
(∀a:elt).(-a)+a=0+;
```

◇

Exemplo 2.3.3 (CELL ([Roş00])). *Foi definida no Exemplo 2.1.7 uma assinatura para especificar o sistema de uma célula de memória de um computador, na qual se podiam escrever e ler dados. O único requisito que se introduz à assinatura é que o valor que se lê na célula é o mesmo que nela se escreveu:*

```
Spec CELL =
[GEN]
    elt;
    cell;
[OP]
    put:elt,cell -> cell;
    get:cell -> elt;
[AX]

(∀e:elt)(∀c:cell)get(put(e,c))=e;
```

Verifica-se facilmente que ambas as Σ_{CELL}-álgebras definidas no Exemplo 2.1.13, são modelos de Cell.

◇

Estas especificações são conhecidas como especificações *flat*. Contudo, quando se pretende trabalhar sistemas reais, reconhece-se rapidamente a necessidade de sistematizar o processo da especificação algébrica, uma vez que, na maior parte dos casos, a sua complexidade implica uma listagem com grande número de fórmulas e resulta em especificações difíceis de trabalhar. Por outro lado, existem sistemas não especificáveis por um conjunto de fórmulas num sistema lógico (ver Observação 2.3.10). É neste contexto que nascem as *linguagens de especificação* e a *especificação estruturada* inspiradas no princípio da *composicionalidade*: constroem-se especificações a partir da composição e extensão de outras especificações mais simples. Este princípio está de acordo com as práticas actuais do *desenvolvimento modular de software*.

Apresenta-se de seguida uma linguagem de especificação com um estrutura bastante simples, por forma a facilitar a generalização dos assuntos trabalhados no texto a outros formalismos mais complexos (ver Observação 2.3.11). A linguagem é constituída por quatro operadores básicos de construção, juntamente com alguns outros definidos à sua custa. Os quatro operadores básicos de construção têm as seguintes funções: um serve para a construção de especificações **flat** (expressões com uma assinatura Σ e um conjunto de fórmulas, denominadas neste contexto por *axiomas*), outro serve para a construção de especificações a partir da combinação de outras duas especificações com a mesma assinatura (**union**), outro serve para a construção de uma especificação de assinatura Σ' a partir de uma especificação sobre Σ e um morfismo $\sigma : \Sigma \to \Sigma'$ (**translate**) e um outro que serve para a partir de uma especificação Σ' e de um

morfismo $\sigma : \Sigma \to \Sigma'$ construir uma especificação em Σ, "preservando"os modelos de SP (**derive**). Nesta linguagem, assumem-se os tipos *Spec*, *Flas*, *Sig*, *Morph*, *Opns* e *sorts* para denotar o conjunto das especificações estruturadas, o conjunto das fórmulas, o conjunto das assinaturas, o conjunto dos morfismos de assinaturas, o conjunto das operações de Σ e o conjunto dos géneros de Σ respectivamente. Segue a definição deste conjunto de operadores:

Definição 2.3.4 (Operadores básicos de construção de especificações estruturadas)**.** *Entendem-se por* operadores básicos de construção de especificações estruturas *os operadores* **flat**, **union**, **translate** *e* **derive**, *definidos da seguinte forma:*

flat

- *Sintaxe:*

  $< ., . >: Sig, Flas \to Spec$

- *Semântica: Sejam Σ uma assinatura heterogénea e Φ um conjunto finito de axiomas.*

  $Sig(< \Sigma, \Phi >) = \Sigma$

  $[\![< \Sigma, \Phi >]\!] =_{def} \{A \in Alg(\Sigma) | A \models \Phi\}$

union

Sejam SP_1 e SP_2 especificações sobre a mesma assinatura:

- *Sintaxe:*

  $< . + . >: Spec, Spec \to Spec$

- *Semântica:*

  $Sig(SP1 + SP2) = Sig(SP_1) = Sig(SP_2)$

  $[\![SP1 + SP2]\!] =_{def} [\![SP_1]\!] \cap [\![SP_2]\!]$

translate

- *Sintaxe:*

translate . **by** . : $Spec, morph \rightarrow Spec$

- *Semântica: Sejam $\sigma : \Sigma \rightarrow \Sigma'$ um morfismo de assinaturas e SP uma especificação estruturada com $Sig(SP) = \Sigma$.*

  $Sig(\textbf{translate } SP \textbf{ by } \sigma) =_{def} \Sigma'$

  $[\![\textbf{translate } SP \textbf{ by } \sigma]\!] =_{def} \{M' \in Alg(\Sigma') | M' \restriction_\sigma \in [\![SP]\!]\}$.

derive

- *Sintaxe:*

  derive from . **by** . : $Spec, morph \rightarrow Spec$

- *Semântica: Sejam $\sigma : \Sigma \rightarrow \Sigma'$ um morfismo de assinaturas e SP uma especificação estruturada com $Sig(SP) = \Sigma'$.*

  $Sig(\textbf{derive from } SP \textbf{ by } \sigma) =_{def} \Sigma$

  $[\![(\textbf{derive from } SP \textbf{ by } \sigma)]\!] =_{def} \{M' \restriction_\sigma \in Alg(\Sigma) | M' \in [\![SP]\!]\}$.

Uma especificação *flat* $SP = \langle \Sigma, \Phi \rangle$ tal que Φ seja um conjunto de equações denomina-se por *especificação equacional*. Existem outros operadores muito úteis no processo de especificação, com são os casos dos operadores **enrich**, **export**, etc.. A razão pela qual se distinguiram os quatro operadores básicos (Definição 2.3.4), deve-se essencialmente ao facto de todos os outros poderem ser construídos pela composição destes quatro:

enrich: É usado quando se pretende enriquecer uma especificação SP com novos géneros, axiomas e operações. Sejam $\Sigma = (S, \Omega)$, $\Sigma' = (S \cup S', \Omega \cup \Omega')$ e $\iota : \Sigma \hookrightarrow \Sigma'$ morfismo de inclusão. Então:

$$\textbf{enrich } SP \textbf{ by sorts } S' \textbf{ opns } F' \textbf{ axioms } \Phi' = (\textbf{translate } SP \textbf{ by } \iota) + \langle \Sigma', \Phi' \rangle$$

export: É um caso particular do operador **translate**, onde o morfismo σ é morfismo ι de inclusão, isto é, para $\iota : \Sigma \hookrightarrow \Sigma'$:

$$\textbf{export } \Sigma' \textbf{ from } SP = \textbf{ derive from } SP \textbf{ by } \iota.$$

São frequentes as especificações cujos modelos são gerados por um conjunto finito de símbolos de funções. Esta característica é formalizada pelo uso de *reachability constraints*:

Definição 2.3.5. reachability constraints

1. *Seja $\Sigma = (S, \Omega)$ uma assinatura heterogénea. Uma* reachability constraint *de Σ é um par $\mathcal{R} = (S_{\mathcal{R}}, \Omega_{\mathcal{R}})$ tal que $\Omega_{\mathcal{R}} \subseteq \Omega$ e*

$$S_{\mathcal{R}} = \{s \in S | \text{ existe um } f \in \Omega_{\mathcal{R}_{\omega,s}}\}.$$

   Um género *$s \in S_{\mathcal{R}}$ denomina-se* género constrained *e um símbolo de função $f \in \Omega_{\mathcal{R}}$ é denominado* símbolo construtor.

2. *Um* termo construtor *é um termo $t \in T_{\Sigma'}(X')_s$, onde $\Sigma' = (S, \Omega_{\mathcal{R}})$, $X'_s = X_s$ se $s \in S \setminus S_{\mathcal{R}}$ e, $X'_s = \emptyset$ se $s \in S_{\mathcal{R}}$. O conjunto dos termos construtores é denotado por $T_{\Sigma,\mathcal{R}}$.*

3. *Uma Σ-álgebra A, satisfaz a* reachability constraint *$\mathcal{R} = (S_{\mathcal{R}}, \Omega_{\mathcal{R}})$ se para qualquer $s \in S$ e para qualquer $a \in A_s$, existe um termo construtor $t \in (T_{\Sigma,\mathcal{R}})_s$ e uma valoração $\alpha : X' \to A$ tal que $I_\alpha(t) = a$. Quando A satisfaz $\mathcal{R}$, escreve-se $A \models \mathcal{R}$.*

Teorema 2.3.6. [Hen97, Facto 2.3.2] *Sejam A uma Σ-álgebra e $\mathcal{R}$ uma* reachability constraint *sobre Σ. As seguintes condições são equivalentes:*

1. *$A \models \mathcal{R}$;*

2. *Para qualquer $s \in S$, para qualquer $a \in A_s$ existe um termo construtor $t \in T_{\Sigma,\mathcal{R}}$ do género s tal que $A, \alpha \models \exists Var(t).x = t$, onde $x \in X_s$, x não pertença a $Var(t)$ e $\alpha : X \to A$ uma valoração tal que $\alpha(x) = a$;*

3. *Para todo $s \in S$,*

$$A \models (\forall x : s) \bigvee_{t \in (T_{\mathcal{R}})_s} \exists Var(t).x \approx t,$$

   onde $\bigvee_{t \in (T_{\mathcal{R}})_s}$ é reunião a (infinita) de todos os termos $t \in (T_{\mathcal{R}})_s$.

Observe-se que qualquer Σ-álgebra satisfaz a *reachability constraint* $(\emptyset, \emptyset)$, uma vez que, neste caso, o conjunto dos termos construtores $T_{\Sigma,\mathcal{R}}$ são os próprios termos T_Σ e, portanto, trivialmente se chega a $A \models \mathcal{R}$. Note-se que o conjunto $S_{\mathcal{R}}$ é unicamente determinado por $\Omega_{\mathcal{R}}$ e, por esta razão, no operador que se define de seguida, considera-se apenas $\Omega_{\mathcal{R}}$ em vez do par $(S_{\mathcal{R}}, \Omega_{\mathcal{R}})$:

Definição 2.3.7. *Define-se o operador* **reach** *da seguinte forma:*

reach

- *Sintaxe:*

$$\textbf{reach}\ .\ \textbf{with}\ . : Spec, Opns \rightarrow Spec$$

- *Semântica:*

  Seja $\mathcal{R} = (S_{\mathcal{R}}, \Omega_{\mathcal{R}})$ *uma* reachability constaint *sobre* $Sig(SP)$.

  $Sig(\textbf{reach}\ SP\ \textbf{with}\ \Omega_{\mathcal{R}}) =_{def} Sig(SP)$

  $[\![\textbf{reach}\ SP\ \textbf{with}\ \Omega_{\mathcal{R}}]\!] =_{def} \{A \in [\![SP]\!] | A \models \mathcal{R}\}$

Um bom exemplo do partido que se pode tirar do operador **enrich**, é a especificação dos números inteiros a partir dos construtores 0 e funções sucessor e predecessor:

Exemplo 2.3.8 (INTZERO)**.** *Considere-se a seguinte especificação dos inteiros definida na Observação 2.3.10. Observe-se que esta especificação é muito livre, na medida em que tanto admite ser modelada pelo conjunto dos números inteiros (com as funções sucessor e antecessor definidas de forma usual), como também pelo conjunto dos reais, etc. O uso de* reachability constraints *restringe a classe de modelos de forma adequada:*

```
Spec INTZERO = reach INT with
[CONS]
    0:-> int;
    s,p: int -> int;
```

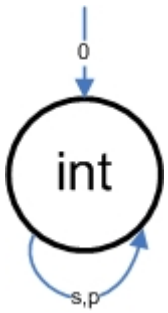

Figura 2.5: Diagrama da assinatura $\Sigma_{INTZERO}$.

◇

Agora, utilizando as especificações anteriores, apresenta-se uma especificação clássica de listas finitas de números inteiros[2].

Exemplo 2.3.9 (INTLIST (Adaptado de [STar]))**.** *Apresenta-se uma especificação das álgebras boolenas:*

```
Spec BOOL =
[GEN]
    bool;
[OP]

true: ->bool;

false: -> bool;

¬:bool -> bool;

∧ :bool,bool -> bool;

=>:bool,bool -> bool;

[AX]

¬(true) = false;

¬(false) = true;

(∀p.bool).p ∧ true=p;

(∀p.bool).p ∧ false=false;

(∀p,q.bool).p => q =¬(p ∧ ¬ q);
```

A expressão que se segue especifica listas de inteiros:

```
Spec INTBOOL = INT + BOOL


Spec INTLIST' =  enrich INTBOOL by
[GEN]
    list
[OP]
```

[2]Por forma a que a leitura da especificação se faça de forma mais intuitiva, usa-se notação *infix* nas operações _ :: _ e _._.

```
     head:list -> int;
     tail:list -> list;
     ∈ :int,list -> bool;
     _._:int,list -> list;
     _::_:list,list -> list;
[AX]
     (∀x:int).(∀l:list)x.l ≠ l;
     (∀x,x':int)(∀l,l':list).x.l=x'.l' → x=x' ∧ l=l';
     (∀x:int)(∀l:list).head(x.l)=x;
     (∀x:int)(∀l:list).tail(x.l)=l;
     (∀l:list).nil::l=l;
     (∀x:int)(∀l,l':list).(x.l)::l'=x.(l::l');
     (∀x:int).x ∈ nil=false;
     (∀x,y:int)(∀l:list).x ≠ y → x  ∈  y.l = x ∈ l;
     (∀x:int)(∀l:list).x ∈ x.l=true;
Spec INTLIST = reach INTLIST' with
[CONS]
    nil:-> list;
    _._:int,list -> list;
    _::_:list,list -> list;
```

◇

Observe-se a importância da utilização do operador **reach** no facto de se pretender especificar listas finitas (uma vez que se requer que estas sejam construídas pelas funções construtoras `_._` e `_::_` a partir da constante `init`), na medida em que a especificação `INTLIST'` especifica listas possivelmente infinitas.

Observação 2.3.10. *Tal como foi referido, algumas especificações estruturadas não são axiomatizáveis por um conjunto de fórmulas, existindo na literatura, algumas caracterizações relativas a esta propriedade. Por exemplo, prova-se que para qualquer*

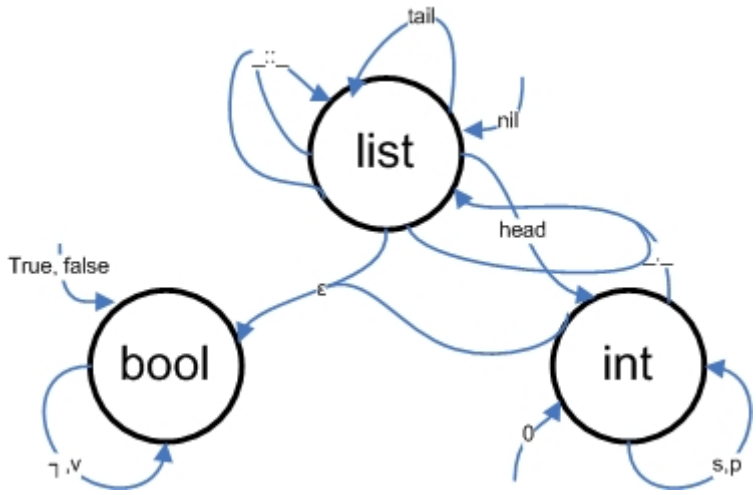

Figura 2.6: Diagrama da assinatura Σ_{LIST}.

especificação SP construída exclusivamente a partir de especificações flat *pelos operadores* **union** *e* **translation**, *existe uma especificação* flat SP' *semanticamente equivalente a SP (cf.* [STar, Tar03]). *Quando se restringe o conjunto de fórmulas de uma assinatura Σ ao conjunto das suas equações, isto é, quando $Flas(\Sigma) = Eq(\Sigma)$, este facto pode ser analisado à luz do* Teorema das Variedades de Birkhoff[3]*, segundo o qual:*

"Uma classe de Σ-álgebras é equacionalmente apresentável sse for fechada por subálgebras, produtos directos e imagens homomorficas."

Assim, quando, por exemplo, se escondem operações (pelo operador **derive**) *segundo um determinado morfismo de inclusão $\sigma : \Sigma \hookrightarrow \Sigma'$ numa determinada especificação $\langle \Sigma, \Phi \rangle$, tem-se que a classe de modelos da especificação* **derive** $\langle \Sigma, \Phi \rangle$ **by** σ *não é geralmente fechada por subálgebras e, por conseguinte, não é axiomatizada por equações. Considere-se, por exemplo, a seguinte especificação algébrica do conjunto dos inteiros:*

```
 Spec INT =
[GEN]
    int
[OP]
    s: int -> int;
```

[3] *Este resultado é um dos mais importantes Teoremas da Álgebra Universal. A sua demonstração para o caso heterogéneo pode ser encontrada em* [MT92, Teorema 5.2.16]*; para o caso homogéneo, reporta-se o leitor para* [Grä79, BS81]*;*

```
    p: int -> int;
    a:     -> int;
    b:     -> int;
[AX]

(∀x:int).s(a)=b;

(∀x:int).p(b)=a;

(∀x:int).s(p(x))=x;

(∀x:int).p(s(x))=x;
```

Tem-se que a $Sig(\texttt{INT})$*-álgebra* A *definida por* $A_{\texttt{int}} = \mathbb{Z}$, $\texttt{s}^A(x) = x+1$ *e* $\texttt{p}^A(x) = x-1$, $a^A = 0$ *e* $b^A = 1$ *é modelo de* INT. *Considere-se agora a especificação*

Spec INT'= derive INT by σ

para σ *definido da seguinte forma:*

$$
\begin{array}{lll}
\sigma: & \Sigma & \rightarrow \; Sig(\texttt{INT}) \\
\sigma_{gen}: & & \\
 & \texttt{int} & \rightarrow \; \texttt{int} \\
\sigma_{op}: & & \\
 & \texttt{a} & \rightarrow \; \texttt{a} \\
 & \texttt{b} & \rightarrow \; \texttt{b} \\
 & \texttt{s} & \rightarrow \; \texttt{s} \\
 & & \rightarrow \; \texttt{p}
\end{array}
$$

Tem-se que a $Sig(\texttt{INT'})$*-álgebra* B *definida por* $B_{\texttt{int}} = \mathbb{N}_0$, $\texttt{s}^B(x) = x + 1$, $a^B = 0$ *e* $b^B = 1$ *é subálgebra de* $A \restriction_\sigma$. *Para que* $B \in [\![\texttt{INT'}]\!]$, *seria necessária a existência de um* $B' \in [\![\texttt{INT}]\!]$ *tal que* $B' \restriction_\sigma = B$. *Para que tal acontecesse, teria que existir uma função* $\texttt{p}^{B'} : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ *tal que juntamente com as funções* $\texttt{s}^{B'} = \texttt{s}^B$, $\texttt{a}^{B'} = \texttt{a}^B$ *e* $\texttt{b}^{B'} = \texttt{b}^B$ *satisfizesse os requisitos de* INT. *Impõe-se na especificação que* s(p(x))=x *e, portanto,* $\texttt{s}^{B'}(\texttt{p}^{B'}\texttt{(x)})$=x, *chegando-se a* $\texttt{p}^{B'}(x) = x - 1$ *que não está definido em* $\texttt{int}^{B'}$. *Prova-se, portanto, que não existe nenhum* B' *tal que* $B' \restriction_\sigma = B$ *e, por conseguinte,* INT *não é fechada por subálgebras. Sai assim do Teorema de* Birkhoff *que não existe nenhuma especificação equacional com a classe de modelos de* INT'.

Observação 2.3.11 (Linguagens de especificação algébrica)**.** *Existem na literatura diversas linguagens e formalismos para a especificação algébrica de software, como, por exemplo, as clássicas ACT ONE* [EFH83]*, ASL* [SW83]*, CLEAR* [BG80] *e EXTEND ML* [ST86]*. Uma linguagem de grande destaque na actualidade é a linguagem CASL (*Common Algebraic specification Language*)* [MHST08] *que resultou do esforço de unificar e standardizar as técnicas e formalismos de uma vasta gama de linguagens de especificação algébrica (incluindo as referidas).*[4] *A linguagem resultante é de grande universalidade, suportando no processo de especificação, por exemplo, o uso de* sub-sorting *e de funções parciais (cf.* [BM04]*). A linguagem apresentada nesta Secção está muito próximo da linguagem ASL. Para uma apresentação retrospectiva de alguns destes formalismos consultar* [BH08, Wir90] *ou* [EM85, EM90] *para a apresentação dos formalismos mais antigos.*

São exemplos de linguagens de especificação orientada a modelos *(não trabalhada no presente texto), as linguagens VDM* [FL98]*, Z* [WD96] *e B* [CM08].

2.4 Processo de Refinamento e Implementação

Um dos principais objectivos de especificar algebricamente um determinado sistema é o de encontrar a sua descrição precisa, de modo a poder implementá-lo de forma clara, exacta e livre de ambiguidades. Neste contexto, o processo de programação traduz-se na construção de uma realização correcta (um programa P) de SP, ou pelo menos numa classe (de $Sig(SP)$-álgebras) $[\![P]\!] \subseteq [\![SP]\!]$ pequena o suficiente para o trabalho desejado.

No processo de implementação de uma componente de software, parte-se de uma especificação inicial SP_0 (descrição inicial do sistema) e faz-se o seu enriquecimento com decisões de implementação (vai-se refinando), por forma a obter uma descrição completa do programa pretendido (álgebra desejada)[5]. Este processo gradual de refinamentos sucessivos é conhecido como o processo de refinamento passo-a-passo (*stepwise refinement*) (cf. [STar, ST97, Mar06], ou [San00] para uma apresentação mais geral do conceito). Naturalmente, o tamanho da classe de modelos da especificação SP vai diminuindo à medida que vai sendo enriquecida com novos requisitos, uma vez que se parte de um caso mais abstracto para um mais concreto. Formalizam-se de seguida

[4] *Este projecto de unificação foi desenvolvido pelo grupo CoFI (ver http://www.cofi.info);*

[5] Em alguns casos esta descrição pode mesmo ser uma única álgebra (a menos de isomorfismos).

estes conceitos:

Definição 2.4.1 (Refinamento)**.** *Sejam SP e SP' duas especificações. SP' é* refinamento *de SP sse:*

1. $Sig(SP) = Sig(SP')$;
2. $[\![SP']\!] \subseteq [\![SP]\!]$;

$SP \rightsquigarrow SP'$ denota que SP' é refinamento *de SP.*

O processo de refinamento passo-a-passo, pode ser entendido como o processo sistemático pelo qual, a partir de uma especificação inicial SP_0, se constroem sucessivamente novas especificações mais restritivas pela introdução de novos requisitos:

$$SP_0 \rightsquigarrow SP_1 \rightsquigarrow SP_2 \rightsquigarrow \cdots \rightsquigarrow SP_{n-1} \rightsquigarrow SP_n,$$

onde para todo $1 \leq i \leq n$, $SP_{i-1} \rightsquigarrow SP_i$ é um passo de refinamento. Os refinamentos $SP_{i-1} \rightsquigarrow SP_i$ são também conhecidas por refinamentos individuais ou por passos de refinamento.

Note-se que se $SP \rightsquigarrow SP'$ e $SP' \rightsquigarrow SP''$ tem-se que $SP \rightsquigarrow SP''$, uma vez que $Sig(SP) = Sig(SP') = Sig(SP'')$ e $[\![SP'']\!] \subseteq [\![SP']\!] \subseteq [\![SP]\!]$. Esta transitividade, conhecida por *composicionalidade vertical*, garante que $SP_0 \rightsquigarrow SP_n$.

Exemplo 2.4.2 (Grupos Abelianos)**.** *Um grupo abeliano (ou comutativo) é todo o grupo que possui operação comutativa. Assim, a classe dos grupos abelianos pode ser obtida pelo enriquecimento de* `GRUPO` *(do Exemplo 2.3.2) por mais esse requisito:*

```
Spec ABEL = enrich GRUPO by
[AX]

(∀a,b:elt).a+b=b+a;
```

Logo `GRUPO` $\rightsquigarrow$ `ABEL`.

Exemplo 2.4.3 (CELL (Adaptado de [Roş00]))**.** *Na especificação* `CELL` *definida no Exemplo 2.3.3, impõem-se apenas que uma célula possua exclusivamente um valor escrito. A especificação que se segue refine* `CELL` *com a imposição de que o seu valor seja o último valor si escrito:*

```
Spec CELL1= enrich CELL  by
[AX]

(∀e,e':elt)(∀c:cell).put(e,put(e',c)))=put(e,c);
```

Tem-se que CELL $\rightsquigarrow$ CELL1*, uma vez que a classe de modelos de* CELL *é restringida por esta nova imposição.*

$\diamond$

Tal como foi referido, durante o processo de refinamento, a especificação a refinar é enriquecida com novos requisitos e decisões de implementação, sendo natural a necessidade de modificar a assinatura da especificação inicial, pela introdução de novos géneros e funções ou mesmo renomeações, etc.. Este papel pode ser interpretado como um morfismo de assinaturas. É com base no *Lema da Satisfação* (Lema 2.2.4), que se apresenta a seguinte generalização do conceito de refinamento:

Definição 2.4.4 (σ-Refinamento)**.** *Seja σ morfismo de assinaturas. Diz-se que SP_1 é σ-refinamento de SP sse:*

1. $Sig(SP_1) = \sigma(Sig(SP))$ *e*
2. $[\![SP_1]\!] \restriction_\sigma \subseteq [\![SP]\!]$,

onde $[\![SP_1]\!] \restriction_\sigma = \{A \restriction_\sigma | A \in [\![SP_1]\!]\}$.

$SP \rightsquigarrow_\sigma SP'$ denota que SP_1 é um σ-refinamento de SP.

Note-se que quando se considera o morfismo identidade id, o conceito de id-refinamento coincide com o conceito de refinamento.

Exemplo 2.4.5. *Apresenta-se de seguida a especificação dos anéis. Um anel é uma álgebra $\langle A, +, * \rangle$ tal que $\langle A, + \rangle$ é grupo abeliano, $\langle A, * \rangle$ é um monóide e $*$ é uma operação associativa e distributiva com $+$. Assim:*

```
Spec ANEL  = enrich ABEL by
[OP]
    * :elt,elt -> elt;
    1:         ->elt;
[AX]

(∀a:elt).(a*1)=a;
```

(∀a:elt).(1*a)=a;

(∀a,b,c:elt).(a*b)*c=a*(b*c);

(∀a,b,c:elt).a*(b+c)=a*b+a*c;

(∀a,b,c:elt).(a+b)*c=a*c+b*c;

A assinatura da especificação ANEL *é resultado da aplicação do seguinte morfismo de assinaturas* $\sigma : Sig(\texttt{ABEL}) \to Sig(\texttt{ANEL})$*:*

$$\begin{array}{lll} \sigma_{gen}: & & \\ & \texttt{elt} & \mapsto \quad \texttt{elt} \\ \sigma_{op}: & & \\ & \texttt{0}_+ & \mapsto \quad \texttt{0}_+ \\ & \texttt{+} & \mapsto \quad \texttt{+} \\ & & \mapsto \quad \texttt{1} \\ & & \mapsto \quad \texttt{*} \end{array}$$

Para toda a $Sig(\texttt{ANEL})$*-álgebra* A*,* $A\restriction_\sigma$ *é a* $Sig(\texttt{ABEL})$*-álgebra seguinte: os géneros definem-se como*

$$(A\restriction_\sigma)_{elt} = A_{\sigma_{gen}(elt)} = A_{elt},$$

e as operações como

$$0_+^{A\restriction_\sigma} = 0_+^A$$

$$+^{A\restriction_\sigma} = +^A,$$

que tal como se pode ver, tem estrutura de grupo. Assim $[\![\texttt{ANEL}]\!]\restriction_\sigma \subseteq [\![\texttt{GRUPOS}]\!]$*, e portanto tem-se que* ANEL *é um* σ*-refinamento de* ABEL*, os seja*

$$\texttt{ABEL} \rightsquigarrow_\sigma \texttt{ANEL}.$$

◇

Exemplo 2.4.6 (CELLNAT)**.** *Suponha-se que se pretende especificar um sistema* CELL1 *(cf. Exemplo 2.4.3) para usar com números naturais. Para tal, considere-se o seguinte morfismo de assinaturas:*

$$
\begin{array}{lll}
\sigma: & Sig(\texttt{CELL1}) & \rightarrow \; Sig(\texttt{CELLNAT}) \\
\sigma_{gen}: & & \\
 & \texttt{elt} & \rightarrow \; \texttt{nat} \\
 & \texttt{cell} & \rightarrow \; \texttt{cell} \\
\sigma_{op}: & & \\
 & \texttt{get} & \rightarrow \; \texttt{get} \\
 & \texttt{put} & \rightarrow \; \texttt{put} \\
 & & \rightarrow \; \texttt{s} \\
 & & \rightarrow \; \texttt{zero}
\end{array}
$$

Assim, por forma a chegar à especificação pretendida pode-se traduzir a especificação CELL1 *por* σ *e depois axiomatizar o conjunto dos números naturais nesta nova especificação. Tem-se por exemplo que* CELL1 $\rightsquigarrow_\sigma$ CELLNAT *para* CELLNAT *definida como se segue:*

```
Spec PRECELLNAT = translate CELL1 by σ

Spec CELLNAT  = enrich PRECELLNAT by
[AX]

(∀x:nat).s(p(x))=x.
```

◇

A propriedade da composicionalidade vertical de σ-refinamentos, sai directamente do resultado do Lema da Satisfação, e do facto da formação de morfismos de assinaturas ser fechada por composição. Assim, se $SP_0 \rightsquigarrow_{\sigma_1} SP_1$ e $SP_1 \rightsquigarrow_{\sigma_2} SP_2$ tem-se que $SP_0 \rightsquigarrow_{\sigma_2 \circ \sigma_1} SP_2$. Para o caso do refinamento passo-a-passo com n passos, tem-se que $SP_0 \rightsquigarrow_{\sigma_n \circ \cdots \circ \sigma_1} SP_n$.

Exemplo 2.4.7. *Tem-se dos exemplos anteriores que,* GRUPO $\rightsquigarrow_{id}$ ABEL*, e* ABEL $\rightsquigarrow_\sigma$ ANEL*. Assim, tem-se que* GRUPO $\rightsquigarrow_{\sigma \circ id}$ ANEL*, ou seja* $[\![\texttt{ANEL}]\!]\restriction_{\sigma \circ id} \subseteq [\![\texttt{GRUPO}]\!]$*, e* $Sig(\texttt{ANEL}) = Sig(\sigma(id(\texttt{GRUPO})))$*.*

Termina-se a Secção com uma caracterização do conceito de σ-refinamento para o caso das especificações *flat*:

Lema 2.4.8. *Sejam* $SP = \langle \Sigma, \Phi \rangle$ *e* $SP' = \langle \Sigma', \Phi' \rangle$ *especificações* flat *e* $\sigma : \Sigma \rightarrow \Sigma'$ *um morfismo de assinaturas. Tem-se que,* $SP \rightsquigarrow_\sigma SP'$ *sse* $SP' \models \sigma(\Phi)$*.*

Demonstração. Suponha-se que $SP \rightsquigarrow_\sigma SP'$. Então, para todo o $A' \in [\![SP']\!]$, $A' \restriction_\sigma \in [\![SP]\!]$, isto é , $A' \restriction_\sigma \models \Phi$. Assim, pelo Lema 2.2.4, $A' \models \sigma(\Phi)$. Por outro lado, tem-se que $SP' \models \sigma(\Phi)$, e portanto, para qualquer $A' \in [\![SP']\!]$, $A' \models \sigma(\Phi)$. Pelo Lema 2.2.4 $A' \restriction_\sigma \models \Phi$, e assim, $A' \restriction_\sigma \in [\![SP]\!]$. Portanto $SP \rightsquigarrow_\sigma SP'$.

□

2.5 Sistemas de Prova

Existem dois momentos principais de verificação no processo de especificação de software: a *validação* da especificação inicial e a *verificação de passos de refinamento*. A validação consiste na verificação da adequação da especificação (inicial) ao sistema a implementar, isto é, na verificação de que os requisitos expressos na especificação são, efectivamente, os requisitos impostos pelo sistema a implementar. São bastante frequentes, no processo de especificação, os fenómenos de *sub-especificação* e *sobre-especificação*. No primeiro, a especificação desenhada é demasiadamente livre, podendo resultar em programas não adequados ao sistema a desenvolver. Na segunda, a especificação desenhada é demasiadamente concreta, podendo descartar modelos admissíveis para a realização do sistema (dificultando a reutilização de componentes de software). A *verificação da correcção de passos de refinamento* está inerente a todo o processo de implementação. Todas a decisões de implementação têm que ser apoiadas por verificações, por forma a garantir a preservação da composicionalidade vertical do processo de refinamento. Deve-se também, em todos estes passos, verificar a *consistência das especificações obtidas*, isto é, verificar se as novas especificações admitem efectivamente modelos. Observe-se que existe uma grande diferença entre estes dois momentos de verificação: enquanto que, no segundo caso, existe uma noção de correcção formal associada ao sistema lógico escolhido para a especificação, no primeiro caso esta referência não existe. Desta forma, para as tarefas de *validação*, recorre-se normalmente aos denominados *"testing-like methods"* [GB99]. Este momento de verificação não é trabalhado no presente texto. Para a verificação da *correcção de passos de refinamento* recorre-se à prova no sistema lógico associado à especificação, sendo portanto natural, o recurso à verificação automática de teoremas, no processo de especificação algébrica.[6]

[6] A algumas linguagens formais de especificação, estão associadas ferramentas automáticas de análise. Por exemplo, à linguagem CASL está associada o conjunto de ferramentas *Heterogeneous Tool Set*-HETS (cf. [BM04], Secção 11) disponível em http://www.cofi.info/tools;

Apresenta-se nesta Secção um sistema de prova correcto e adequado para a verificação de propriedades em especificações desenhadas na linguagem definida na Secção 2.3.

Especificações Flat

Quando se pretende verificar uma equação numa especificação *flat* recorre-se, frequentemente, ao cálculo equacional:

Definição 2.5.1 (Cálculo equacional)**.** *Uma Σ-fórmula é consequência sintáctica de uma especificação* **flat** $SP = \langle \Sigma, \Phi \rangle$*, se pode ser derivada pela aplicação das seguintes regras de inferência:*

$$\frac{}{SP \vdash_\Sigma (\forall X).t = t'} \text{ para todo } (\forall X).t = t' \in \Phi;$$

$$\frac{}{SP \vdash_\Sigma (\forall X).t = t} \text{ para todo } t \in T_\Sigma(X);$$

$$\frac{SP \vdash_\Sigma (\forall X).t' = t}{SP \vdash_\Sigma (\forall X).t = t'};$$

$$\frac{SP \vdash_\Sigma (\forall X).t = t' \text{ e } SP \vdash_\Sigma (\forall X).t' = t''}{SP \vdash_\Sigma (\forall X).t = t''};$$

$$\frac{SP \vdash_\Sigma (\forall X).t_1 = t'_1 \text{ e } SP \vdash_\Sigma (\forall X).t_n = t'_n}{SP \vdash_\Sigma (\forall X).f(t_1, \ldots, t_n) = f(t'_1, \ldots, t'_n)}$$

para toda $f : s_1, \ldots, s_n \to s \in \Sigma$ e $t_i, t'_i \in T_\Sigma(X)_{s_i}, 1 \leq i \leq n$;

$$\frac{SP \vdash_\Sigma (\forall X).t = t'}{SP \vdash_\Sigma (\forall Y).I_\alpha(t) = I_\alpha(t')}$$

para toda $\alpha : X \to T_\Sigma(Y)$;

O Teorema que se segue estabelece a adequação e completude do cálculo apresentado:

Teorema 2.5.2 (Teorema da completude e adequação do cálculo equacional)**.** *Sejam $SP = \langle \Sigma, \Phi \rangle$ uma especificação* flat *e $(\forall X).t{=}t'$ uma equação. Tem-se que:*

$$SP \vdash_\Sigma (\forall X).t{=}t' \text{ sse } SP \models (\forall X).t{=}t'.$$

A demonstração do resultado pode ser encontrada em [STar]. A implicação " $\Rightarrow$ " aparece como *Teorema da adequação do cálculo equacional* [7](Teorema 2.4.6), e a implicação " $\Leftarrow$ " como *Teorema da completude cálculo equacional* (Teorema 2.4.11).

Especificações Estruturadas

O cálculo em cima apresentado não trabalha a verificação de especificações estruturadas construídas à custa de operadores de construção (Secção 2.3) que não o **flat** aplicado a conjuntos de equações. Para esta verificação, existem na literatura dois tipos de sistemas de prova: os *sistemas não composicionais* e os *sistemas composicionais*. Por princípio, os sistemas de prova não composicionais transformam uma especificação estruturada mais complexa numa outra "mais simples", com os mesmos teoremas. Assim, o problema de verificar uma propriedade numa especificação é reduzido à verificação standard num conjunto adequado de axiomas (ver Observação 2.3.10). Um dos sistemas de prova baseado neste princípio é o da *abordagem da forma normal* (cf. [BCH99, Hen97]).

Nos sistemas composicionais a verificação de uma fórmula numa especificação é feita nas suas "componentes ou sub-especificações", seguindo regras impostas pela natureza dos operadores utilizados na sua construção. Desta forma, há a possibilidade de reutilização de demonstrações existentes nas suas componentes, indo-se assim, ao encontro da filosofia da especificação estruturada e, por conseguinte, do desenvolvimento modular de software.

O sistema de prova que se apresenta de seguida é um sistema de prova composicional apresentado em [Bor99]:

Definição 2.5.3 (Sistema de prova para especificações estruturadas)**.**

$$\frac{\phi \in \Phi}{\langle \Sigma, \Phi \rangle \vdash \phi}$$

$$\frac{SP_1 \vdash \phi}{SP_1 + SP_2 \vdash \phi}$$

$$\frac{SP_2 \vdash \phi}{SP_1 + SP_2 \vdash \phi}$$

$$\frac{SP \vdash \phi}{\textbf{translate } SP \textbf{ by } \sigma \vdash \sigma(\phi)}$$

[7]No texto *Soundness of equational calculus*;

$$\frac{SP' \vdash \sigma(\phi)}{\textbf{derive from } SP' \textbf{ by } \sigma \vdash \phi}$$

$$\frac{SP \vdash \phi_i \textit{ para todo o } i \in I \textit{ e } \langle \Sigma, \{\phi_i | i \in I\}\rangle \vdash \phi}{SP \vdash \phi}$$

Existem algumas variantes deste cálculo na literatura adequadas a outras escolhas de operadores básicos de construção (cf. [ST97, Hen97, BCH99]).

O Teorema que se segue estabelece a completude e a adequação do sistema de prova definido:

Teorema 2.5.4. *Sejam SP uma especificação estruturada de assinatura Σ e ϕ uma Σ-fórmula. Tem-se que:*

$$SP \models \phi \textit{ sse } SP \vdash \phi.$$

A demonstração deste resultado é feita em [Bor99].

2.6 Sistemas de Reescrita

A forma mais natural de verificar a igualdade entre dois objectos é a de simplificar cada um o mais possível segundo um conjunto definido de regras, (por exemplo, na aritmética o uso das regras de prioridade da multiplicação) e depois compará-los entre si. Formaliza-se nesta Secção este processo para a comparação entre termos de uma assinatura Σ. Todos os resultados e conceitos que se aqui se apresentam (confluência, forma normal, terminação etc.), podem ser estudados e trabalhados numa estrutura abstracta denominada por *sistema abstracto de reescrita* (cf. [Der05]) de forma independente da natureza dos objectos a reescrever. Este assunto pode ser encontrado para o caso dos sistemas de reescrita de termos sobre assinaturas heterogéneas, por exemplo, em [EM85, DJ90, GM96a, STar]. Em [Der05] trabalha-se o caso dos sistemas de reescrita abstractos, particularizando-se, de seguida, esses mesmos resultados a um grande número de sistemas.

Definição 2.6.1 (Regra e sistema de reescrita)**.** *Seja $\Sigma = (S, \Omega)$ uma assinatura heterogénea. Uma* regra de reescrita *r é uma expressão $u_1 \rhd u_2$ tal que $u_1, u_2 \in T_\Sigma(X)_s$ para algum género $s \in S$. Entende-se por* sistema de reescrita *um conjunto de regras de reescrita. Muitas vezes denota-se a regra $r = u_1 \rhd u_2$ simplesmente por $\rhd_r$.*

Dado um conjunto de equações Φ sobre uma assinatura Σ entende-se por *sistema de reescrita induzido por* Φ o sistema $R(\Phi) = \{t_1 \rhd t_2 | t_1 = t_2 \in \Phi\}$, e por *conjunto de equações do sistema*, o conjunto de equações $E(R) = \{t_1 = t_2 | t_1 \rhd t_2 \in R\}$. Observe-se que para todo o conjunto de equações Φ, $E(R(\Phi)) = \Phi$ e para todo o sistema de reescrita R, $R(E(R)) = R$. Dada uma especificação equacional $SP = \langle \Sigma, \Phi \rangle$ denota-se $R(\Phi)$ por $R(SP)$.

Exemplo 2.6.2 (FLAGS)**.** *O sistema de reescrita induzido pelos axiomas da especificação* `FLAGS` *do Exemplo 3.1.10 é constituído pelas seguintes regras de reescrita:*

`?up(up(x))` $\rhd$ `true`

`?up(down(x))` $\rhd$ `false`

`?up(rev(x))` $\rhd \neg$`(?up(x))`

$\diamond$

Definição 2.6.3 (Redução por reescrita)**.** *Sejam $\Sigma = (S, \Omega)$ assinatura heterogénea, t_1 e $t_2 \in T_\Sigma(X)_s$ dois termos do género $s \in S$ e R um sistema de reescrita sobre Σ. Diz-se que t_1 se* reduz num só passo *a t_2 em R por r e escreve-se $t_1 \rhd_r t_2$ sse existe uma regra de reescrita $r = u_1 \rhd u_2 \in R$ e existe uma substituição $\alpha : X \to T_\Sigma(X)$ tais que:*

1. *$\alpha(u_1)$ é subtermo de t_1;*
2. *t_2 é obtido de t_1 pela substituição de $\alpha(u_1)$ por $\alpha(u_2)$.*

Um termo t diz-se irredutível *por R, caso não exista nenhuma regra $r = u_1 \rhd u_2 \in R$ nem nenhuma $\alpha : X \to T_\Sigma(X)$ tal que $\alpha(u_1)$ seja subtermo de t_1.*

Tem-se desta forma que uma regra $r = u_1 \rhd u_2$ do sistema R gera uma relação binária $\rhd_r \subseteq T_\Sigma(X) \times T_\Sigma(X)$ denominada por *redução por um passo gerada por r*, sendo a relação $\rhd_R = \bigcup_{r \in R} \rhd_r$ denominada por *redução por um passo gerada por R*. O fecho transitivo da relação $\rhd_R$ denota-se por $\rhd_R^*$, e é denominado por redução em R.

Definição 2.6.4 (Computação terminada e divergente)**.** *Sejam Σ uma assinatura heterogénea e R um sistema de reescrita sobre Σ. Entende-se por:*

- computação terminada *toda a sequência $t_1, \ldots, t_n$ de termos de Σ tal que para todo o $i \in \{1, \ldots, n-1\}$ $t_i \rhd_R t_{i+1}$ e t_n seja irredutível por R.*

- computação divergente *se existe uma sucessão* $t_1, t_2, \ldots$ *tal que* $t_i \rhd_R t_{i+1}$ *para todo o* $i \geq 1$.

Exemplo 2.6.5. *Considerem-se as seguintes regras de reescrita:*

`f(x)`$\rhd$`g(x)`

`g(x)`$\rhd$`f(x)`

É evidente que não existem computações terminantes neste sistema de reescrita.

$\diamond$

Definição 2.6.6 (Forma normal e derivação por reescrita)**.** *Sejam* Σ *assinatura heterogénea,* $t, t' \in T_\Sigma(X)_s$ *e* R *sistema de reescrita sobre* Σ*. Diz-se que* t' é a forma normal de t*, e escreve-se* $t \blacktriangleright_R t'$ *sse existe uma computação terminada* $t_1, \ldots, t_n$ *tal que* $t = t_1$ *e* $t' = t_n$*. Uma equação* t_1=t_2 deriva-se por reescrita *em* R*, e escreve-se* $\vdash_R t_1$=t_2*, se existe um termo* t_3 *tal que* $t_1 \blacktriangleright_R t_3$ *e* $t_2 \blacktriangleright_R t_3$*, isto é, quando partilham a mesma forma normal.*

As definições que se seguem são de grande importância para a caracterização de um sistema de reescrita:

Definição 2.6.7 (Sistemas de reescrita terminantes, confluentes e completos)**.** *Um sistema de reescrita diz-se* terminante *se não permite computações divergentes; diz-se* confluente *se existe um* t_3 *tal que* $t_1 \rhd_R^* t_3$ *e* $t_2 \rhd_R^* t_3$ *sempre que* $t \rhd_R^* t_1$ *e* $t \rhd_R^* t_2$*; diz-se* completo *se for confluente e terminante.*

Tal como foi referido na introdução da Secção, a comparação de dois termos $t_1, t_2 \in T_\Sigma(X)_s$ é feita pela comparação das suas formas normais. Contudo, para que tal seja feito correctamente, é necessária a unicidade da forma normal de um termo. A proposição que se segue caracteriza essa propriedade:

Proposição 2.6.8 (Sistema de reescrita completo)**.** *Num sistema de reescrita completo todo o termo tem uma única forma normal.*

O Teorema que se segue estabelece a relação entre um sistema de reescrita e o cálculo equacional (Definição 2.5.1):

Teorema 2.6.9. *Seja* $SP = \langle \Sigma, \Phi \rangle$ *uma especificação* flat *tal que* $R(SP)$ *é um sistema de reescrita completo. Tem-se que:*

$$R(SP) \vdash_R t_1 = t_2 \text{ sse } SP \models t_1 = t_2.$$

A demonstração do resultado anterior pode ser encontrada em [STar, Teoremas 2.6.8 e 2.6.10] (para as implicações " $\Rightarrow$ " e " $\Leftarrow$ " respectivamente).

Capítulo 3

Igualdade Comportamental

Apresentaram-se no Capítulo 1 algumas motivações para a adaptação dos conceitos apresentados no Capítulo anterior à especificação algébrica de programas com dados encapsulados. Apresenta-se agora um exemplo ilustrativo destas motivações: considere-se o caso da implementação de uma *interface* online de uma conta bancária, onde o utilizador dispõe das operações `saldo` e `pagamento` para consultar o seu saldo e efectuar pagamentos respectivamente (adaptado de [BH99]). Tem-se, desta forma, que o estado de duas contas só pode ser comparado pelo resultado de operações nelas efectuadas. Por outro lado, não é relevante para o utilizador a forma da implementação interna do género de dados `conta`, podendo, por exemplo, ser implementados sob a forma do conjunto dos inteiros ou, caso se pretenda guardar o histórico de movimentos efectuados na conta (ver Exemplo 2.1.13), sob a forma de listas de inteiros (a função `saldo` retorna o primeiro elemento da lista). Observe-se também que os comportamentos observáveis do programa (pelo utilizador ou por outros módulos) perante ambas as implementações são exactamente os mesmos (no caso da interface ser constituída apenas pelas operações mencionadas). Por outro lado, dois elementos do género `conta` podem responder da mesma forma perante todas as computações, não sendo realmente iguais. Por exemplo, no caso de se escolher a segunda hipótese de implementação, os elementos $c_1 = [a, b :: \omega]$ e $c_2 = [a, c :: \omega']$ sendo diferentes, comportam-se da mesma forma perante todas as computações

$$\texttt{saldo}(c), \texttt{saldo(pagamento}(c)), \ldots, \texttt{saldo(pagamento(}\cdots\texttt{(pagamento}(c))))\cdots,$$

havendo assim uma "espécie de igualdade"entre estes dois elementos, na medida em que não se distinguem via computações. Compreende-se, desta forma, que a relação de igualdade estrita é demasiadamente forte para o tratamento semântico de sistemas

deste tipo. Estas duas equivalências vão ser trabalhadas no texto pelos conceitos de equivalência observacional (entre álgebras) e igualdade observacional (entre elementos).

Existem duas abordagens ao tema na literatura: uma baseada na *equivalência (comportamental/observacional) entre álgebras*, denominada por *abordagem Abstractor* e outra baseada na *relação de satisfação comportamental/observacional* entre elementos (Definição 3.1.2), sendo esta, a abordagem central do texto.

3.1 Igualdade Comportamental

A relação de igualdade mostra-se muitas vezes demasiadamente forte para a análise de sistemas de software com dados encapsulados (Secção 3). Apresenta-se, neste Capítulo, como que uma adaptação dos conceitos trabalhados na lógica (validade, satisfação, etc.) mais adequado ao tratamento semântico de programas deste tipo. Para o efeito, numa primeira fase (nesta Secção), estuda-se a substituição da relação de igualdade estrita por uma qualquer congruência parcial (Definição 2.1.15) e, numa segunda (Secção 3.1.1), por forma a conseguir a semântica procurada, particularizam-se estes resultados ao caso da congruência parcial *igualdade observacional*.

Definição 3.1.1 (Igualdade comportamental e comportamento)**.** *Seja* $\Sigma = (\Sigma, \Omega)$ *uma assinatura heterogénea. Uma* Σ-igualdade comportamental $\approx$ *é uma* S*-família de* Σ*-congruências parciais* $(\approx^A)_{A \in Alg(\Sigma)}$ *. Para qualquer* Σ*-álgebra* A*, define-se o* comportamento de A por $\approx^A$*, como sendo a álgebra quociente de* $Dom(\approx^A)$ *por* $\approx^A$*. O comportamento de* A *por* $\approx^A$ *denota-se por* $A/\approx^A$*.*

Para uma classe de Σ-álgebras C, $C/\approx$ denota a classe de Σ-álgebras $\{A/\approx^A \mid A \in C\}$, e dada uma especificação SP, $SP/\approx$ denota a classe $[\![SP]\!]/\approx$. O comportamento de A é também conhecido como a *"black box view"* da álgebra A. A analogia da álgebra quociente $A/\approx^A$ com uma caixa negra, deve-se ao facto de $A/\approx^A$ identificar os elementos comportamentalmente iguais, ou seja, que são indistinguíveis "de fora da caixa". Uma outra importante característica de $A/\approx^A$ é o facto dos elementos irrelevantes serem "esquecidos"(elementos não pertencentes a $Dom(\approx^A)$ não são considerados).

Define-se, de seguida, a relação de satisfação baseada no conceito da igualdade comportamental:

Definição 3.1.2 (Relação de satisfação comportamental)**.** *Sejam* Σ *uma assinatura e* $\approx = (\approx^A)_{A \in Alg(\Sigma)}$ *uma relação de* Σ*-igualdade comportamental. Define-se a relação*

$\models_{\approx}$, *denominada por* relação de satisfação comportamental*, da seguinte forma:*

Sejam A uma Σ-álgebra, $l, r \in T_\Sigma(X)_s$ dois termos do género s, ϕ e ψ duas Σ-fórmulas, $\{\phi_i | i \in I\}$ um conjunto contável de Σ-fórmulas e $\alpha : X \to Dom(\approx^A)$ uma função de valoração.

1. $A, \alpha \models_{\approx} l{=}r$ *se* $I_\alpha(l) \approx^A I_\alpha(r)$ *se verifica;*
2. $A, \alpha \models_{\approx} \neg\phi$ *se* $A, \alpha \models_{\approx} \phi$ *não se verifica;*
3. $A, \alpha \models_{\approx} \phi \wedge \psi$ *se* $A, \alpha \models_{\approx} \phi$ *e* $A, \alpha \models_{\approx} \psi$;
4. $A, \alpha \models_{\approx} (\forall x : s).\phi$ *se para qualquer função de valoração* $\beta : X \to Dom(\approx^A)$ *tal que* $\beta(y) = \alpha(y)$*, para todos* $y \neq x$ *se tem que* $A, \beta \models_{\approx} \phi$;
5. $A, \alpha \models_{\approx} \bigwedge_{i \in I} \phi_i$ *verifica-se sse para todo o* $i \in I$ *se tem* $A, \alpha \models_{\approx} \phi_i$.

Escreve-se $A \models_{\approx} \phi$ quando para toda a valoração $\alpha : X \to Dom(\approx^A)$ se tem que $A, \alpha \models_{\approx} \phi$.

A relação definida pode ser vista como uma generalização da relação de satisfação *Tarskiana* na medida em que a relação de igualdade estrita é a relação de igualdade comportamental $\models_{=}$.

Dadas uma classe de Σ-álgebras C, uma Σ-fórmula ϕ, e uma Σ-igualdade comportamental $\approx$, escreve-se $C \models_{\approx} \phi$ se para toda a Σ-álgebra $A \in C$, $A \models_{\approx} \phi$. Dada uma especificação algébrica SP de assinatura Σ, escreve-se $SP \models_{\approx} \phi$ em vez de $[\![SP]\!] \models_{\approx} \phi$. Dados dois termos $t, t' \in T_\Sigma(X)$, a expressão $t \approx^A t'$ significa que $A \models_{\approx} t = t'$, e a expressão $t \approx_\Sigma t'$ (ou simplesmente $t \approx t'$) significa que para toda a Σ-álgebra A, $A \models_{\approx} t = t'$.

Definição 3.1.3 (Teoria comportamental)**.** *Sejam Σ uma assinatura, $\approx$ uma Σ-igualdade comportamental e $C \subseteq Alg(\Sigma)$ uma classe de Σ-álgebras. Define-se,* teoria comportamental de C por $\approx$*, em símbolos, $Th_\approx(C)$, da seguinte forma:*

$$Th_\approx(C) =_{def} \{\phi \in Flas(\Sigma) | C \models_{\approx} \phi\}.$$

O Teorema que se segue, estabelece uma conexão entre as relações de satisfação standard e de satisfação comportamental:

Teorema 3.1.4. *Sejam Σ uma assinatura e $\approx$ uma Σ-igualdade comportamental. Para toda Σ-álgebra A, toda Σ-fórmula ϕ e para toda a classe de Σ-álgebras C, tem-se que:*

1. $A \models_{\approx} \phi$ *sse* $A/\approx^A \models \phi$;

2. $SP \models_{\approx} \phi$ *sse* $SP/\approx \models \phi$;

3. $Th_{\approx}(C) = Th(C/\approx)$, *onde*

$$C/\approx = \{A/\approx^A \mid A \in C\}.$$

Demonstração. A demonstração do Teorema tem por base o seguinte resultado:

Lema 3.1.5. [Hen97, Teorema 3.3.6] *Seja* Σ *assinatura heterogénea e* A *uma* Σ*-álgebra. Para toda a* Σ*-fórmula* ϕ *e para toda a valoração* $\alpha : X \to Dom(\approx^A)$, *tem-se que* $A, \alpha \models_{\approx} \phi$ *sse* $A/\approx^A, \pi \circ \alpha \models \phi$, *onde* $\pi : Dom(\approx^A) \to A/\approx^A$ *representa o epimorfismo canónico.*

Sejam A e ϕ tais que $A \models_{\approx} \phi$, e $\beta : X \to A/\approx^A$ uma função de valoração arbitrária. Uma vez que π é epimorfismo canónico, tem-se que, existe uma valoração $\alpha : X \to Dom(\approx^A)$ tal que $\beta = \pi \circ \alpha$. Tem-se também que $A, \alpha \models_{\approx} \phi$ (uma vez que por hipótese $A \models_{\approx} \phi$), e pelo Lema 3.1.5 $A/\approx^A, \pi \circ \alpha \models \phi$, isto é, $A/\approx^A, \beta \models \phi$. Uma vez que β arbitrário, tem-se $A/\approx^A \models \phi$ tal como se pretendia $(*)$.

Sejam agora A, $\approx^A$ e ϕ tais que $A/\approx^A \models \phi$, isto é, tais que para toda a valoração $\beta : X \to Dom(A)$, $A/\approx^A, \beta \models \phi$. Tem-se assim que para toda a valoração $\alpha : X \to A$, $A/\approx^A, \alpha \circ \pi \models \phi$ e pelo Lema 3.1.5 $A, \alpha \models_{\approx} \phi$. Uma vez que α é valoração arbitrária, tem-se que $A \models_{\approx} \phi$ tal como se pretendia $(**)$.

Por (*) e (**) fica provado 1.. Os pontos 2. e 3. saem directamente de 1.. □

Assim, pelo Teorema anterior, verificar a validade comportamental de uma fórmula ϕ numa Σ-álgebra A, é a mesma coisa que verificar a validade (standard) de ϕ no seu comportamento. Contudo, quando se pretende trabalhar ao nível das classe de Σ-álgebras, o significado não é o mesmo. O facto da formação de quocientes não preservar, no caso geral, a validade de fórmulas arbitrárias, diminui a potencial utilidade prática do resultado, uma vez que, mesmo que a classe $\mathbf{C}$ seja axiomatizável, não há métodos de prova para trabalhar a teoria standard $\mathbf{C}/\approx$ (cf. [BH96]). Outras caracterizações das teorias comportamentais vão ser trabalhadas no seguimento da texto (Secção 3.5.1).

Apresenta-se, de seguida, um exemplo de uma igualdade comportamental:

Exemplo 3.1.6 (Adaptado de [BBK95])**.** *Considere-se a especificação* `INT` *definida na Observação 2.3.10, e a* Σ_{INT}*-álgebra* A *representada na figura (os círculos representam os elementos de* $A_{\texttt{int}}$*, e as setas representam a aplicação das funções* $\texttt{s}^A$ *e* $\texttt{p}^A$*).*

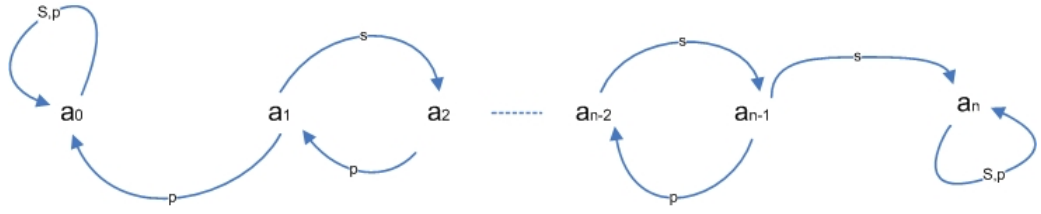

Figura 3.1: Uma Σ_{INT}-álgebra.

É evidente que a Σ_{INT}-álgebra representada na figura não é modelo de `INT`*, uma vez que por exemplo, para $\alpha : X \to A$ tal que $\alpha_{\texttt{int}}(x) = a_1$, tem-se que $I_\alpha(\texttt{s(p}(x)) = a_0$, $I_\alpha(x) = a_1$, falhando assim o axioma (2)(o axioma (1) também falha para o caso de $\alpha(x) = a_{n-1}$).*

Considere-se a Σ_{INT}-igualdade comportamental:

$$\approx^A = \{(a_i, a_i) |\ \textit{para todo o } i \textit{ tal que } 2 \leq i \leq n-2\} \cup \{\langle a_0, a_1\rangle, \langle a_{n-1}, a_n\rangle\},$$

onde $\langle x, y\rangle$ representa a congruência total gerada por x e y.

Tem-se agora que A é modelo comportamental de `INT` *segundo $\approx^A$, uma vez que para $\alpha(x) = a_1$, $I_\alpha(\texttt{s(p}(x)) \approx^A I_\alpha(x)$ (e o mesmo acontece para $\alpha(x) = a_{n-1}$).*

◇

Outros exemplos interessantes deste tipo de igualdades podem ser encontrados na literatura (cf. [Hen97]); contudo, a igualdade comportamental mais estudada é a *igualdade observacional* que se apresenta na Secção que se segue.

É, por vezes, necessário considerar sistemas onde só variáveis de determinados géneros são tidas como variáveis de Input. Efectivamente, na prática da especificação algébrica de sistemas, tal restrição surge de forma natural, na medida em que faz todo o sentido considerar apenas os valores alcançados pelo sistema via possíveis computações, isto é, via perturbações do sistema por input. Por forma a modelar esta situação, distinguem-se na assinatura da especificação do sistema um conjunto de géneros denominado por *géneros de input do sistema*. Assim, para uma assinatura Σ define-se o conjunto das *variáveis de Input* X_{In}, como sendo o conjunto de variáveis para Σ definido da seguinte forma: $(X_{In})_s$ é um conjunto de variáveis não vazio para $s \in In$ e $(X_{In})_s = \emptyset$ no caso $s \in S \setminus In$. Para toda a Σ-álgebra A, $A[X_{In}]$ denota a menor subálgebra de A "gerada por X_{In}"definida da seguinte forma: $A[X_{In}]_s = \{a \in A_s |$ existe um termo $t \in T_\Sigma(X_{In})_s$ e uma valoração $\alpha : X_{In} \to A$ tal que $I_\alpha(t) = a\}$.

A Σ-álgebra $A[X_{In}]$ denomina-se por *subálgebra de A gerada pelo input In*. Observe-se que a álgebra $A[X_{In}]$, admite exactamente os valores alcançáveis do sistema via computações admissíveis. A assinatura Σ diz-se *sensível* a In se para todo $s \in S \setminus In$, o conjunto de termos $T_\Sigma(X_{In})_s$ é não vazio, isto é, se para todo $s \in S \setminus In$ existe pelo menos um termo t do género s, construído por símbolos de função de Ω e (possivelmente) por variáveis de $X_{s'}$, com $s' \in In$.

3.1.1 O Caso Observacional

Tal como foi referido (Secção 3), os géneros de uma assinatura podem ser divididos em géneros observáveis e géneros não observáveis. Esta divisão está na base da igualdade observacional. Os géneros observáveis são também conhecidos por géneros visíveis (*visible sorts*) e os não observáveis por géneros escondidos (*hidden sorts*). Nas representações gráficas os géneros não observáveis aparecem muitas vezes em sombreado por forma a que se distingam dos géneros observáveis. Apresentam-se de seguida alguns exemplos representativos deste tipo de assinaturas:

Exemplo 3.1.7 (Autómatos)**.** *Considerem-se os alfabetos de Input* `X` *e de Output* `B` *como géneros observáveis e o género dos estados* `Z` *como um género não observável:*

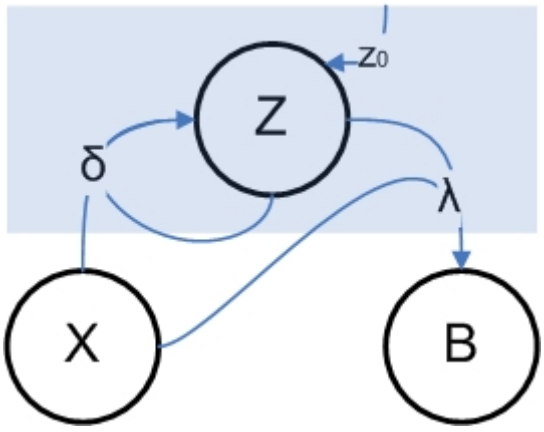

Figura 3.2: Diagrama de $\Sigma_{AUT-I/O}$.

◇

Exemplo 3.1.8 (CELL ([Roş00]))**.** *Considere-se o caso da célula de memória apresentado no Exemplo 2.4.3. Naturalmente, o utilizador só tem acesso ao valor que*

está na célula via `get`. *A assinatura* $\Sigma_{\texttt{cell}}$ *com o conjunto de géneros observáveis* $Obs = \{\texttt{elt}\}$ *especifica esta situação:*

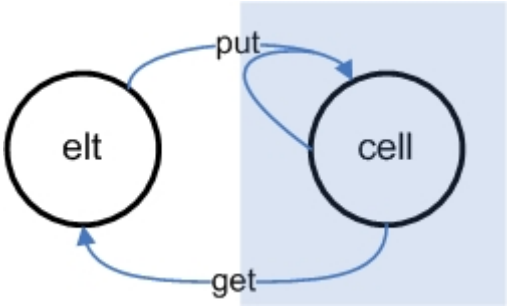

Figura 3.3: Diagrama de Σ_{CELL}.

Exemplo 3.1.9 (STATE)**.** *Considere-se agora a especificação* `STATE` *descrita no Exemplo 2.1.8. Considerando os géneros* $\{\texttt{elt}, \texttt{id}\}$ *como os géneros observáveis, o género* $\{\texttt{state}\}$ *representa o estado das células com o qual se pode interagir a partir das funções* `lookup` *e* `update` *da seguinte forma: para atribuir a um valor* `id` *de* `ID` *um valor* `elt`*, usa-se a função* `update`*, enquanto que para consultar o valor de um determinado* `id`*, usa-se a função* `lookup`*. Voltando à analogia do disco duro, a função* `update` *escreve o valor* `elt` *na célula com morada* `id` *e, a função* `lookup` *vai ler o valor* `elt` *da célula com morada* `id`*. Seja* `ID` *uma especificação de identificadores de moradas (por exemplo a especificação* `INT` *definida na Observação 2.3.10).*

```
Spec STATE' = enrich ID by
[GEN]
    state;
    elt;
[OBS]
    id;
    elt;
[OP]
    lookup: id,state -> elt;
    init: -> state;
    update:id,elt,state -> state;
[AX]
```

```
(∀i:id).lookup(i,init)=0;

(∀n:elt)(∀i:id)(∀s:state).lookup(i,update(i,n,s))=n;

(∀n:elt)(∀i,j:id)(∀s:state).

    [i ≠ j → lookup(i,update(j,n,s))=lookup(i,s)];

(∀n,m:elt)(∀i,j:id)(∀s:state).

   update(i,n,update(j,m,s))=update(i,n,s);

(∀n,m:elt)(∀i,j:id)(∀s:state).

    [i ≠ j → update(i,n,update(j,m,s))=update(j,m,update(i,n,s))];

Spec STATE = reach STATE' by
[CONS]
    init: -> state;
    update:id,elt,state -> state;
```

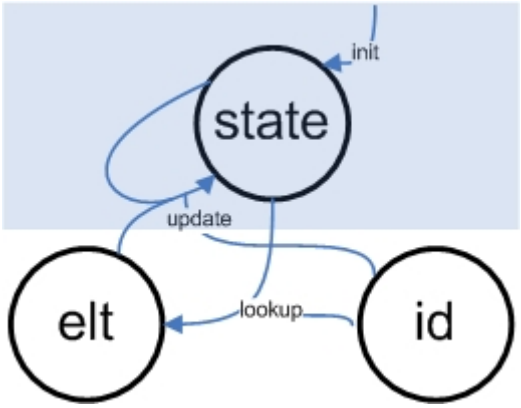

Figura 3.4: Diagrama de Σ_{STATE}.

◇

Exemplo 3.1.10 (FLAGS ([GM99])). *Outro exemplo clássico da área é o exemplo* Flags *que consiste num método de escalonamento de recurso em sistemas partilhados (na gestão de processadores, impressoras, em redes, etc...). O método funciona da seguinte forma: a cada recurso está associado uma bandeira (*flag*),*[1] *que toma o valor*

[1] *O uso do símbolo da bandeira está relacionado com os antigos semáforos de transito;*

"`up`*"quando está a ser utilizado por um processo, voltando a "*`down`*"quando está novamente disponível. Quando a* `flag` *está* `up` *o acesso ao esse recurso é negado. A única forma de conhecer a disponibilidade desse recurso é através da função* `?up` *de resultado booleano. Tem-se então que* $Obs = \{\texttt{bool}\}$*, e:*

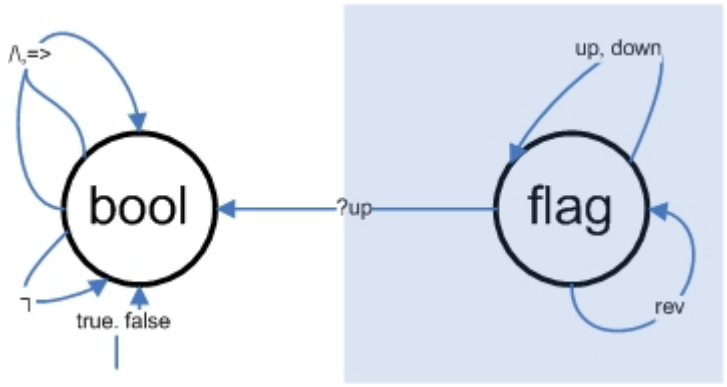

Figura 3.5: Diagrama de Σ_{FLAGS}*.*

```
Spec FLAGS = enrich BOOL by
[GEN]
    flag;
[OBS]
    bool;
[OP]
    up: flag -> flag;
    down: flag -> flag;
    rev: flag -> flag;
    ?up: flag -> bool;
[Axiomas]
```

(∀x:flag)?up(up(x))=true;

(∀x:flag)?up(down(x))=false;

(∀x:flag)?up(rev(x))=not(?up(x));

Onde a especificação `BOOL` *representa a especificação usual das álgebras boolenas (Exemplo 2.3.9).*

◇

Exemplo 3.1.11 (STREAM). *Uma outra importante estrutura infinita de dados é a estrutura* `Stream`*, que consiste numa lista (infinita) de elementos, cujo o primeiro elemento é consultável pela operação* `head`*. As operações* `tail` *e* `_::_` *retiram e introduzem um elemento no início da lista:*

```
Spec STREAM =
[GEN]
    stream;
    elt;
[OBS]
    elt;
[OP]
    head:stream -> elt;
    tail:stream -> stream;
    _::_ :elt ,stream -> stream;
[AX]
(∀s:stream,x:elt).head(e::x)=e;
(∀s:stream,x:elt).tail(e :: x)=x;
```

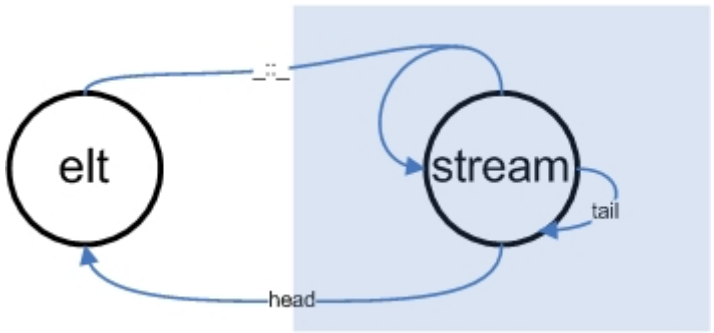

Figura 3.6: Diagrama de Σ_{STREAM}.

◇

Tal como foi sugerido na Secção 3, perante a abordagem observacional, dois elementos são considerados observacionalmente iguais se não se destingem quando submetidos a experiências de resultado observável. No contexto da análise de sistemas, tais experiências podem ser compreendidas como computações de resultado visível e são formalizadas algebricamente a partir do conceito de *contexto observável*:

Definição 3.1.12 (Contextos e Contextos Observáveis). *Sejam $\Sigma = (S, \Omega)$ uma assinatura, $Obs \subseteq S$ um conjunto de géneros observáveis, X um conjuntos de variáveis para Σ e $Z = (\{z_s\})_{s \in S}$ uma S-família de conjuntos singulares. Um* s-contexto *de Σ é um termo $c \in T_\Sigma(X \cup \{z_s\})_{s'}$, onde a variável z_s é denominada por* variável contextual de c. *No caso de $s' \in Obs$, um s-contexto denomina-se por* s-contexto observável (relativamente ao conjunto Obs) *de Σ. O conjunto dos s-contextos de Σ denota-se por $\mathcal{C}_\Sigma(s)$ e o conjunto dos Σ-contextos observáveis por $\mathcal{C}_\Sigma^{Obs}$.*

O conjunto das variáveis não contextuais de um contexto c denota-se por $Var(c)$, e $c[t]$ denota o termo obtido pela substituição da variável contextual z_s de c pelo termo $t \in T_\Sigma(X)_s$. A notação e a terminologia apresentadas nesta definição variam bastante de autor para autor. Por exemplo, nos trabalhos de *J. Goguen* (e seus seguidores), os contextos observáveis são conhecidos como "*expiriments*"e são denotados por ε_Σ, o conjunto $\mathcal{C}(s)$ aparece como $\mathcal{C}[\bullet{:}s]$, e uma variável contextual z_s aparece como $\bullet : s$.

Exemplo 3.1.13 (Autómatos). *O conjunto dos contextos observáveis de $\mathcal{C}^{Obs}_{\Sigma_{auto-I/O}}$ é o conjunto de termos $\{\lambda(\delta(\delta(\cdots(\delta(z_Z, x_1)))\cdots)x_{n-1}), x_n) | x_1, \ldots, x_n \in X, n \in \mathbb{N}\} \cup \{z_{\mathtt{B}}, z_{\mathtt{X}}\}$.*

◇

Exemplo 3.1.14 (FLAGS). *Para a assinatura Σ utilizada na especificação* `FLAGS` *são exemplos de contextos observáveis de Σ_{FLAGS} os elementos do conjunto*

$$\{\mathtt{not}\}^\star(\mathtt{up?}(\{\mathtt{up}, \mathtt{down}, \mathtt{rev}\}^\star(z_{flag}))).$$

◇

É com base no conceito de contexto que se define *igualdade parcial contextual* e *igualdade parcial observacional*:

Definição 3.1.15 (Igualdade Parcial Contextual). *Sejam $\Sigma = (S, \Omega)$ uma assinatura heterogénea, $In \subseteq S$ o conjunto de géneros de* input *tal que Σ sensível a In, $\mathcal{C}$ um conjunto arbitrário de Σ-contextos e A uma Σ-álgebra. Dois elementos $a, b \in A_s$ dizem-se* contextualmente iguais segundo $\mathcal{C}$ e In *se e só se $a, b \in A[X_{In}]_s$ e, para todo o contexto $c \in \mathcal{C}(s)$, para todas as valorações $\alpha, \beta : X \cup \{Z_s\} \to A[X_{In}]$ tais que $\alpha(x) = \beta(x)$ para todo o $x \in X$ e, $\alpha(z_s) = a$ e $\beta(z_s) = b$, tem-se que $I_\alpha(c) = I_\beta(c)$. Quando a e b são contextualmente iguais segundo $\mathcal{C}$ e In,*

escreve-se $a \approx^A_{\mathcal{C},In} b$.

Definição 3.1.16 (Igualdade Observacional Parcial e Total)**.** *Sejam $\Sigma = (S, \Omega)$ uma assinatura, A uma Σ-álgebra, $Obs \subseteq S$ o conjunto de géneros observáveis e $In \subseteq S$ conjunto de géneros de input tal que Σ-sensível a In. A igualdade parcial em A induzida por $\mathcal{C}_\Sigma^{Obs}$ e In denomina-se* igualdade observacional parcial*, e denota-se por $\approx^A_{Obs,In}$. No caso em que $In = S$ a relação definida diz-se* igualdade observacional total *ou simplesmente* igualdade observacional*. Normalmente escreve-se $\approx^A_{Obs}$ em vez de $\approx^A_{Obs,S}$.*

O Teorema que se segue, particulariza a Σ-igualdade observacional (parcial e total), como casos de Σ-igualdades comportamentais:

Teorema 3.1.17. *A igualdade observacional $\approx^A_{Obs}$ é uma congruência total em A e a igualdade observacional $\approx^A_{Obs,In}$ é uma Σ-congruência parcial em A de domínio $A[X_{In}]$.*

Demonstração. A demonstração do Teorema baseia-se na seguinte caracterização da relação de congruência:

Lema 3.1.18. *Sejam $\Sigma = (S, \Omega)$ uma assinatura heterogénea, A uma Σ-álgebra, $f \in \Omega_{s_1 \cdots s_n, s}$, $(\theta_s)_{s \in S}$ uma S-relação binária em A e para todo $i \in \{1, \ldots, n\}$, $a_i, b_i \in A_i$ elementos tais que $a_i \theta_{s_i} b_i$. A relação θ é compatível com f sse para todo o $i \in \{1, \ldots, n\}$ se tem que para todas as valorações $\alpha, \beta : X \to A$, tais que $\alpha(x) = \beta(x)$ para todo o $x_{s_j} \in X_{s_j}$ $j \neq i$ e, tais que $\alpha(x_{s_i}) = a_i$ e $\beta(x_{s_i}) = b_i$ se tem que $I_\alpha(f(x_1, \ldots, x_n)) \approx_s I_\beta(f(x_1, \ldots, x_n))$.*

(Caso $In = S$): considerem-se para um $i \in \{1, \ldots, n\}$ os elementos $a_i, b_i \in A_{s_i}$ tais que $a_i (\approx^A_{Obs})_{s_i} b_i$. Observe-se que a transitividade e simetria de $\approx^A_{Obs,In}$ saem directamente da Definição 3.1.1, na medida em que dadas estas hipóteses, tem-se que $I_\alpha(c) = I_\beta(c) \Rightarrow I_\beta(c) = I_\alpha(c)$, e que dado um elemento $d_i \in A_{s_i}$ tal que para toda a valoração $\gamma : X \cup \{Z_{s_i}\} \to A$ com $\beta(x) = \gamma(x)$ para todo $x \in X$ e que $\gamma(z_s) = d_i$ se tem que $I_\alpha(c) = I_\gamma(c)$, tem-se que $a_i \approx^A_{Obs,In} d_i$. Pretende-se agora provar a compatibilidade de $\approx^A_{Obs,In}$ em Σ (pelo Lema 3.1.18). Por um lado, tem-se que para todo o $c' \in \mathcal{C}_\Sigma^{Obs}(s)$, $c'[f(\bar{x}, z_{s_i})] \in \mathcal{C}_\Sigma^{Obs}(s_i)$, onde $f(\bar{x}, z_{s_i})$ abrevia a expressão $f(x_1, \ldots, x_{i-1}, z_{s_i}, x_{i+1}, \ldots, x_n)$. Para o caso em que $\alpha(x) = \alpha'(x)$ para todo $x \in X$ e $\alpha(z_{s_i}) = a_i$ e $\alpha'(z_s) = f(\bar{x}, a_i))$ tem-se que $I_{\alpha'}(c') = I_{\alpha'}(c'(f(\bar{x}, z_{s_i}))$, e pela mesma razão, $I_{\beta'}(c') = I_{\beta'}(c'[f(\bar{x}, z_{s_i}])$. Por outro lado, tem-se por hipótese que $I_\alpha(c) = I_\beta(c)$ para todo o contexto $c \in \mathcal{C}_\Sigma^{Obs}(s_i)$ donde $I_{\alpha'}(c') = I_{\beta'}(c')$, tal como se pretendia.

(Caso $In \subseteq S$): a demonstração do caso parcial sai de forma idêntica à da demonstração anterior, com a diferença que as funções de valoração α e β têm conjuntos de chegada $A[X_{In}]$ em vez de A.

□

A um modelo comportamental de uma especificação SP a respeito de uma igualdade observacional $\approx_{Obs}$ denomina-se *modelo observacional de SP a respeito a Obs* ou, sempre que Obs for claro no contexto, simplesmente por *modelo observacional de SP*. Desta forma, todos os resultados estudados para o caso da relação de igualdade comportamental podem ser aplicados no estudo da relação de igualdade observacional.

Exemplo 3.1.19 (Autómatos)**.** *O conceito de* equivalência de estados num autómato, *está na base dos bem conhecidos algoritmos de construção de autómatos reduzidos* (*cf.* [HUM01, DW02])*. Dois estados z_1 e z'_1 de um autómato H dizem-se equivalentes $z_1 \sim z'_1$ sse para qualquer palavra $\omega \in X^*$, $\widehat{\lambda}(z_1, \omega) = \widehat{\lambda}(z'_1, \omega)$, isto é, se para toda a palavra $a_1 \cdots a_n \in X^*$,*

$$\lambda(z_1, a_1) \cdots \lambda(z_n, a_n) = \lambda(z'_1, a_1) \cdots \lambda(z'_n, a_n),$$

onde $z_i = \delta(z_{i-1}, a_{i-1})$, $2 \leq i \leq n$ e $z'_i = \delta(z'_{i-1}, a_{i-1})$, $2 \leq i \leq n$.

Tem-se portanto que $z_1 \sim z'_1$ sse

$$\textit{para todo } a_1 \in X, \lambda(z_1, a_1) = \lambda(z'_1, a_1), \ \textit{e}$$

$$\textit{para todo } a_1, a_2 \in X, \lambda(\delta(z_1, a_1), a_2) = \lambda(\delta(z'_1, a_1), a_2), \ \textit{e}$$

$$\textit{para todo } a_1, a_2, a_3 \in X, \lambda(\delta(\delta(z_1, a_1), a_2), a_3) = \lambda(\delta(\delta(z'_1, a_1), a_2), a_3), \ \textit{e}$$

$$\vdots$$

pelo que, sai do Exemplo 3.1.13 que, para todo $c \in \mathcal{C}^{Obs}_{\Sigma_{aut-I/O}}$, para todas a valorações $\alpha_1, \alpha_2 : X \cup Z \to H$ tais que $\alpha_1(x) = \alpha_2(x)$ se $x \in X$ e $\alpha_1(z_Z) = z_1$ e $\alpha_2(z_Z) = z'_1$, tem-se que $I_{\alpha_1}(c) = I_{\alpha_2}(c)$. Fica provado que a igualdade entre estados de um autómato não é mais do que a Σ-igualdade observacional ($\sim = \approx^H_{Obs}$).

◇

Exemplo 3.1.20 (CELL)**.** *Considere-se a especificação* `CELL1` *apresentada no Exemplo 2.4.3 e a Sig(`CELL`)-álgebra B definida no Exemplo 2.1.13. Tem-se que B não é modelo estrito da especificação* `CELL1`*, na medida em que, para qualquer $\omega \in \mathbb{N}^*$ e quaisquer* `e,e'` $\in \mathbb{N}$ *se tem que* $\texttt{put}^B(\texttt{e},\texttt{put}^B(\texttt{e'},\omega)) = \texttt{ee'}\omega$ *e que* $\texttt{e}\omega = \texttt{put}^B(\texttt{e},\omega)$*. No entanto, facilmente se verifica que* `B` $\models_{Obs}$ `put(e,put(e',x))=put(e',x)`*. Tem-se desta forma que a álgebra B é modelo observacional de* `CELL1` *não o sendo no sentido estrito.*

Apresentam-se, de seguida, dois lemas que estão na base de grande parte dos métodos conhecidos de verificação de propriedades observacionais (nomeadamente nos descritos nas Secções 3.5.1 e 4.1).

Teorema 3.1.21. *Sejam A uma Σ-álgebra e $\approx^A$ uma Σ-congruência parcial em A. Se $Dom(\approx^A) = Dom(\approx^A_{Obs,In})$, e se para todos os géneros $s \in Obs$, $(\approx^A)_s$ coincide com a igualdade, tem-se se $a \approx^A b$ então $a \approx^A_{Obs,In} b$.*

Demonstração. Sejam $a, b \in A[X_{In}]$, tais que $a \approx^A b$. Tem-se por hipótese que $\approx^A$ é relação de congruência parcial, pelo que, para todo o Σ-contexto $\mathcal{C}_\Sigma$, para todas as valorações $\alpha, \beta : X \to A[X_{In}]$ tais que $\alpha(x) = \beta(x)$ se $x \in X$, e $\alpha(z_s) = a$, $\beta(z_s) = b$ se tem que $I_\alpha(z_s) \approx^A I_\beta(z_s)$ (pela propriedade da compatibilidade da Σ-congruência parcial). Tem-se também por hipótese que para todo o $s \in Obs$, a relação $\approx^A$ coincide com a relação $=$, e portanto, em particular para todos os contextos observáveis $c \in \mathcal{C}_\Sigma^{Obs}$ $I_\alpha(c) = I_\beta(c)$. Logo, pela definição de Σ-igualdade observacional, tem-se que $a \approx^A_{Obs,In} b$, tal como se pretendia demonstrar.

□

As congruências $\approx$ nas condições do Lema 3.1.21 aparecem na literatura denominadas por congruências escondidas *(hidden congruence)*. Este resultado que caracteriza a Igualdade Observacional como congruência escondida maximal, está na base da aplicação de métodos co-indutivos para a verificação de propriedades observacionais (Secção 4.1). O Teorema que se segue, estabelece uma caracterização da Σ-igualdade observacional $\approx^A_{Obs,in}$ a partir de uma Σ-igualdade contextual $\approx^A_{\mathcal{C},In}$.

Teorema 3.1.22 (Caracterização da igualdade observacional por igualdades contextuais)**.** *Sejam $\Sigma = (S, \Omega)$ assinatura, $Obs \subseteq S$ conjunto de géneros observáveis, A uma Σ-álgebra e $\mathcal{C} \subseteq \mathcal{C}_\Sigma^{Obs}$ conjunto arbitrário de Σ-contextos observáveis, tal que, para todo $s \in Obs$, $z_s \in \mathcal{C}$. Então, tem-se que $\approx^A_{\mathcal{C},In} = \approx^A_{Obs,In}$ sse $\approx^A_{\mathcal{C},In}$ for uma Σ-congruência parcial.*

Demonstração. Por um lado, tem-se que $\approx^A_{\mathcal{C},In}$ é Σ-congruência parcial, uma vez que, pelo Teorema 3.1.17, $\approx^A_{Obs,In}$ também o é.

Por outro lado, considere-se a Σ-congruência parcial $\approx^A_{\mathcal{C},In}$. Uma vez que por hipótese $\mathcal{C} \subseteq \mathcal{C}_\Sigma^{Obs}$, tem-se pela definição de Σ-igualdade contextual (Definição 3.1.15) que $\approx^A_{Obs,In} \subseteq \approx^A_{\mathcal{C},In}$ (1). Por outro lado, tem-se por hipótese que para todo o género $s \in S$, $z_s \in \mathcal{C}$, e portanto, pela definição de Σ-igualdade contextual (Definição 3.1.15),

tem-se que $\approx^A_{\mathcal{C},In}$ coincide com a relação de igualdade $=$ nos géneros observáveis. Uma vez que por hipótese se tem que $\approx^A_{\mathcal{C},In}$ é relação de Σ-congruência parcial, sai do Lema 3.1.21 que $\approx^A_{\mathcal{C},In} \subseteq \approx^A_{Obs,In}$ (2). De (1) e (2) fica provado que $\approx^A_{\mathcal{C},In} = \approx^A_{Obs,In}$, tal como se pretendia. □

Um conjunto de contextos indicado em alguns trabalhos (cf. [BH94, BH96, BH98]) como sendo uma boa aproximação de um conjunto com as propriedades do Teorema 3.1.22, é o denominado *conjunto dos contextos cruciais*, que consiste no menor conjunto de contextos $\mathcal{C} \subseteq \mathcal{C}^{Obs}_{\Sigma}$ com as seguintes propriedades: para todo o $s \in Obs$, $z_s \in \mathcal{C}$ e, para toda a função $f : s_1, \ldots, s_n \rightarrow s \in \Sigma$, para todo o $k \in \{1, \ldots, n\}$ tal que $s_k \in S \setminus Obs$, $f(x_1, \ldots, x_{k-1}, z_{s_k}, x_{k+1}, \ldots, x_n) \in \mathcal{C}$.

3.2 Operador Comportamental

Nesta Secção, considera-se um enriquecimento da linguagem de especificação algébrica apresentada na Secção 2.3, por um operador de construção de especificações estruturadas, denominado por *operador comportamental*. O objectivo de tal enriquecimento é o de encontrar uma forma de trabalhar os conceitos apresentados anteriormente em especificações estruturadas arbitrárias. Considere-se o seguinte operador de classes de Σ-álgebras:

Definição 3.2.1 (Classe comportamental)**.** *Sejam* $\approx = (\approx^A)_{A \in Alg(\Sigma)}$ *uma* Σ*-igualdade comportamental e* $C \subseteq Alg(\Sigma)$ *uma classe de* Σ*-álgebras. Define-se* classe comportamental de C *por* $\approx$ *como sendo a classe*

$$Beh_{\approx}(C) =_{def} \{A \in Alg(\Sigma) | A/\approx^A \in C\}.$$

Definição 3.2.2 (Igualdade Comportamental Isomorficamente Compatível)**.** *Uma* Σ*-igualdade comportamental* $\approx = (\approx^A)_{A \in Alg(\Sigma)}$ *diz-se* isomorficamente compatível *se para quaisquer duas* Σ*-álgebras* A *e* B *isomorfas, se tem que* $A/\approx^A$ *e* $B/\approx^B$ *também são isomorfas.*

A construção *classe comportamental* induz um novo operador de construção de especificações estruturadas: o operador **behaviour.wrt.**. Para este trabalho, define-se *BehEq* como sendo o tipo de todas as Σ-igualdades comportamentais isomorficamente compatíveis:

behaviour

- Sintaxe:

$$\textbf{behaviour.wrt}. : Spec, BehEq \rightarrow Spec$$

- Semântica:

$$Sig(\textbf{behaviour } SP \textbf{ wrt } \approx) =_{def} Sig(SP)$$

$$[\![\textbf{behaviour } SP \textbf{ wrt } \approx]\!] =_{def} Beh_{\approx}([\![SP]\!])$$

Este operador tem um papel central na formalização da noção de implementação comportamental a partir da qual, uma implementação é vista como comportamentalmente correcta se os seus comportamentos satisfazem as propriedades requeridas pela especificação SP.

Assim, a classe de modelos $[\![\textbf{behaviour } SP \textbf{ wrt } \approx]\!]$ corresponde à classe de todas as Σ-álgebras cujos comportamentos pertencem a $[\![SP]\!]$, ou seja, o operador **behaviour** SP **wrt** $\approx$ especifica a classe de Σ-álgebras dos *"comportamentos desejados"*, que pode ser entendida como a classe das *"realizações comportamentalmente correctas"* de SP.

O Teorema que se segue ilustra a universalidade da abordagem comportamental em especificações estruturadas. A partir dele, mostra-se que abordagens que trabalham exclusivamente com especificações flat (caso das abordagens apresentadas no Capítulo 4) podem ser encarados como casos específicos da abordagem apresentada neste Capítulo.

Teorema 3.2.3. *Seja $SP = \langle \Sigma, \Phi \rangle$ uma especificação flat e seja $\approx$ uma Σ-igualdade comportamental. Então:*

$$[\![\textbf{\textit{behaviour}} \ SP \ \textbf{\textit{wrt}} \approx]\!] = \{A \in Alg(\Sigma) | A \models_{\approx} \Phi\}.$$

Demonstração. Tem-se por definição do operador **behaviour** que para qualquer Σ-álgebra A, $A \in [\![\textbf{behaviour } SP \textbf{ wrt } \approx]\!]$ se e só se $A/\approx^A \in [\![SP]\!]$. Uma vez que SP é especificação *flat* tem-se que $A/\approx^A \in [\![SP]\!]$ se e só se $A/\approx^A \models \Phi$, e portanto, sai do Teorema 3.1.4 que $A \models_{\approx} \Phi$ tal com se pretendia. □

Observe-se que existem fórmulas e igualdades comportamentais tais que $A \models \phi$ e $A \not\models_{\approx} \phi$ e, por conseguinte, especificações SP tais que $[\![SP]\!] \not\subseteq [\![\textbf{behaviour } SP \textbf{ by } \approx]\!]$:

Exemplo 3.2.4 ([HN94])**.** *Considere-se a seguinte especificação:*

```
Spec DEMO =
[GEN]
    s;
    s';
[OBS]
    s;
[OP]
    a:   -> s;
    b:   -> s;
    c:   -> s';
    d:   -> s';
    f: s'-> s;
[AX]

f(c)=f(d);
c=d → a=b;
```

Considere-se agora uma Σ_{DEMO}*-álgebra* A *tal que* $\mathtt{a}^A \neq \mathtt{b}^A$*, que* $\mathtt{c}^A \neq \mathtt{d}^A$ *e que* $\mathtt{f(c)}^A = \mathtt{f(d)}^A$*. Tem-se que* A *é um modelo estrito da especificação não o sendo em sentido observacional, na medida em que satisfaz estritamente o segundo axioma não o satisfazendo observacionalmente. Tem-se portanto que* $[\![\mathtt{DEMO}]\!] \not\subseteq [\![\textbf{\textit{behaviour}}\ \mathtt{DEMO}\ \textbf{\textit{by}}\ \approx_s]\!]$.

Uma Σ-fórmula ϕ diz-se $\approx$*-compatível* se para toda a Σ-álgebra A e para toda a valoração $\alpha : X \to A$, $A, \alpha \models \phi \Rightarrow A, \alpha \models_{\approx} \phi$. Uma especificação SP diz-se *comportamentalmente fechada a respeito de* $\approx$ se e só se $[\![SP]\!] \subseteq [\![\textbf{behaviour}\ SP\ \textbf{by}\ \approx]\!]$, isto é, se $[\![SP]\!]/\approx \subseteq [\![SP]\!]$. Quando a igualdade comportamental é uma igualdade observacional diz-se que a especificação é *observacionalmente fechada*.

O teorema que se segue caracteriza esta propriedade em especificações *flat*:

Teorema 3.2.5. *[Hen97, Teorema 3.5.4] Sejam* $SP = \langle \Sigma, \Phi \rangle$ *uma especificação flat e* $\approx$ *uma* Σ*-igualdade comportamental. São equivalentes as seguintes afirmações:*

1. SP *é comportamentalmente fechada a respeito de* $\approx$*;*
2. *todos os* $\phi \in \Phi$ *são* $\approx$*-compatíveis;*
3. $SP \models_{\approx} \phi$*, para todos os* $\phi \in \Phi$*.*

Existem na literatura algumas caracterizações relativas à $\approx_{Obs}$-compatibilidade de fórmulas. Segundo [Hen97, Exemplo 3.3.11], para que uma Σ-fórmula ϕ seja $\approx_{Obs}$-compatível é suficiente (mas não necessário) que possa ser representada sob a forma

$$(Q_1 x_1 : s_1) \cdots (Q_n x_n : s_n).\ \bigwedge_{1 \leq i \leq k} ((\bigvee_{1 \leq j \leq m} p_{ij} \neq q_{ij}) \vee (\bigvee_{1 \leq k \leq m} t_{ij} = r_{ij})),$$

para $Q_i \in \{\forall, \exists\}$ e $p_{ij}, q_{ij}, t_{ij}, r_{ij} \in T_\Sigma(X)$ tais que p_{ij}, q_{ij} termos observáveis. Uma Σ-fórmula é $\approx_{Obs,In}$-compatível se for representável como em cima e se possuir apenas, como variáveis universalmente quantificadas, variáveis de Input.

Pelo Teorema 3.2.5 tem-se que são exemplos de especificações observacionalmente fechadas as especificações equacionais e as especificações flat axiomatizadas exclusivamente por equações condicionais com premissas observáveis, na medida em que se pode representar $t_1 = t'_1 \wedge \cdots \wedge t_n = t'_n \rightarrow t = t'$ na forma $(\bigvee_{1 \leq i \leq n} t_i \neq t'_i) \wedge t = t'$. Tem-se ainda que toda a especificação construída a partir dos operadores básicos de construção apresentados na Definição 2.3.4 com excepção do operador **derive** e tal que todos os axiomas das suas especificações constituintes sejam representáveis como em cima é observacionalmente fechada (cf. [Hen97]).

3.3 Especificações *Abstractor*

Nas secções anteriores, apresentou-se o estudo da semântica observacional de especificações algébricas via relação de igualdade observacional $\approx^A_{Obs}$. Apresenta-se, de seguida, de forma sucinta, a outra grande abordagem da literatura relativa a este assunto: a abordagem à semântica observacional de especificações algébricas via *relação de equivalência entre álgebras* $\equiv_{Obs}$, também conhecida por abordagem *abstractor*. Enquanto que na primeira, a classe de modelos de uma especificação SP, é determinada pela relação de satisfação comportamental $\models_\approx$, definida à custa da relação de igualdade observacional $\approx$ (por todas as $Sig(SP)$-álgebras que satisfaçam os requisitos de SP segundo $\approx_{Obs}$), esta segunda é determinada à custa do estudo do fecho da relação de equivalência $\equiv_{Obs}$ entre $Sig(SP)$-álgebras. Esta abordagem é adoptada por vários autores tais como *H. Reichel*, *D. Sannella*, *A. Tarlecki* (cf. [Rei85, STar]). Em [BHW95] faz-se a ponte entre as estas duas abordagens.

Definição 3.3.1 (Relação de equivalência isomorficamente protegida)**.** *Uma relação de equivalência* $\equiv \subseteq Alg(\Sigma) \times Alg(\Sigma)$ *diz-se protegida por isomorfismos quando se tem*

que:

$$\textit{Se } A \textit{ e } B \textit{ são isomorfos, então } A \equiv B.$$

Definição 3.3.2. *Sejam* $\equiv \subseteq Alg(\Sigma) \times Alg(\Sigma)$ *uma relação de equivalência entre* Σ*-álgebras isomorficamente protegida e* C *uma classe de* Σ*-álgebras. Define-se a classe de* Σ*-álgebras* $Abs_{\equiv}(C)$ *da seguinte forma:*

$$Abs_{\equiv}(C) = \{A \in Alg(\Sigma) | A \equiv B \textit{ para algum } B \in C\}.$$

Esta construção induz a definição de um operador de construção de especificações estruturadas:

abstractor

- Sintaxe:

$$\textbf{abstractor}.\ \textbf{wrt}\ . : Spec, equiv \rightarrow Spec$$

- Semântica:

$$Sig(\textbf{abstractor}\ SP\ \textbf{wrt}\ \equiv) =_{def} Sig(SP)$$

$$[\![\textbf{abstractor}\ SP\ \textbf{wrt}\ \equiv]\!] =_{def} Abs_{\equiv}([\![SP]\!])$$

onde *equiv* representa o conjunto de todas as relações de equivalência isomorficamente protegidas em $Alg(Sig(SP))$.

O conceito que se segue estabelece uma ligação entre as abordagens à observabilidade via igualdade observacional entre elementos e via equivalência observacional entre álgebras:

Definição 3.3.3 (Relação factorizável)**.** *A relação* $\equiv$ *diz-se factorizável por* $\approx$ *quando se tem:*

$$A \equiv B \textit{ sse } A/\approx^A \cong B/\approx^B .$$

Diz-se simplesmente factorizável quando existe uma relação $\approx$ *com estas propriedades.*

O Caso Observacional

Tal como se referiu na Secção 3, dois programas (com a mesma *interface*) são equivalentes observacionalmente se responderem da mesma forma a todas as computações. É este o principio formalizado nesta Secção:

Definição 3.3.4. *Sejam $\Sigma = (S, \Omega)$ assinatura, $Obs \subseteq S$ conjunto de géneros observáveis e $In \subseteq S$ conjunto de géneros de input sensível a Σ. As Σ-álgebras A e B dizem-se* observacionalmente equivalentes *$A \equiv_{obs,In} B$, se existe uma S-família $Y_{In} = (Y_{In})_{s \in S}$, tal que $(Y_{In})_s = Y_s$ se $s \in In$ e $(Y_{In})_s = \emptyset$ caso contrário, e duas valorações $\alpha_1 : Y_{In} \to A$ e $\beta_1 : Y_{In} \to B$ tais que para todo $s \in In$, $(\alpha_1)_s : (Y_{In})_S \to A$ e $(\beta_1)_s : (Y_{In})_S \to B$ sobrejectivas, e para todos os termos $t, r \in T_\Sigma(Y_{In})_s$ com $s \in Obs$ se tem que:*

$$I_{\alpha_1}(t) = I_{\alpha_1}(r) \text{ sse } I_{\beta_1}(t) = I_{\beta_1}(r).$$

Existem abordagens à observabilidade mais restritivas do que a que se apresenta no texto, nomeadamente, as que fixam as interpretações dos géneros observáveis. Esta abordagem denominada por *fixed-data semantics* é, por exemplo, adaptada em [STar, GM00] entre muitos outros. Efectivamente, esta restrição faz bastante sentido no estudo semântico de objectos de software, na medida em que é natural a imposição da natureza dos aspectos externos do objecto, nomeadamente a natureza dos seus géneros de *output*, por forma a que seja garantida a sua compatibilidade com outros objectos do sistema (ver Secção 3). A Definição 3.3.4 adaptada a esta abordagem é muito mais intuitiva relativamente aos propósitos descritos no início da Secção, na medida em que, a compatibilidade entre os universos observáveis de ambas as Σ-álgebras ($A_s = B_s$ para todo o $s \in Obs$), faz com que duas Σ-álgebras se considerem observacionalmente equivalentes segundo o mesmo conjunto de géneros observáveis $Obs \subseteq S$, se todos os Σ-termos observáveis tomarem os mesmos valores em ambas as álgebras, isto é, se responderem com o mesmo resultado perante as mesmas computações (cf. [STar, Definição 8.2.2]).

O Teorema que se segue estabelece a ligação entre a abordagem *abstractor* e a apresentada nas secções anteriores. A demonstração do Teorema pode ser encontrada em [BHW95, Exemplo 5.4.].

Teorema 3.3.5. *Sejam $\Sigma = (S, F)$ uma assinatura, $Obs \subseteq S$ conjunto de géneros observáveis e $In \subseteq S$ conjunto de géneros de Input tal Σ sensível a In. A equivalência observacional $\equiv_{Obs,In}$: $Alg(\Sigma) \times Alg(\Sigma)$ é factorizável pela Σ-igualdade observacional $\approx_{Obs,In}$.*

Teorema 3.3.6. *Uma especificação SP é observacionalmente fechada a respeito a Obs sse:*

$$[\![\textbf{\textit{behaviour}}\ SP\ \textbf{\textit{wrt}}\ \approx_{Obs}]\!] =_{spec} [\![\textbf{\textit{abstract}}\ SP\ \textbf{\textit{wrt}}\ \equiv_{Obs}]\!].$$

A demonstração do resultado pode ser encontrada em [STar, Teorema 8.3.29]. Em particular, tem-se que para toda a especificação SP equacional ou axiomatizada por um conjunto de equações condicionais com premissas observáveis,

$$[\![\textbf{behaviour } SP \textbf{ wrt } \approx_{Obs}]\!] =_{spec} [\![\textbf{abstract } SP \textbf{ wrt } \equiv_{Obs}]\!].$$

Exemplo 3.3.7 (Autómato minimal e equivalência de autómatos)**.** *Considere-se novamente a assinatura dos autómatos com output introduzida no Exemplo 2.1.6. Um autómato $H \in Alg(\Sigma_{AUT-I/O})$ diz-se* reduzido*, se para todos o estados $z_1, z_2 \in Z^H$, se $z_1 \sim z_2$, então $z_1 = z_2$. Prova-se no âmbito da teoria da computação que para todo o autómato H, existe um autómato reduzido H' tal que $L(H) = L(H')$ (cf.* [HUM01]*). Do Exemplo 3.1.19, sai que H é reduzido, se para todos os estados $z_1, z_2 \in Z^H$, se $z_1 \approx^H_{Obs} z_2$, então $z_1 = z_2$ e, portanto, o autómato em questão é autómato comportamento $H/\approx^H_{Obs}$ (Definição 3.1.4).*

Outro assunto de interesse da teoria da computação é o de verificar se dois autómato H_1 e H_2 são equivalentes (observacionalmente). Do Teorema 3.3.5, tem-se que a equivalência observacional $\equiv_{Obs}$ é factorizável pela igualdade observacional $\approx$ e, portanto, quando se pretende verificar se dois autómatos são equivalentes (no sentido de aceitarem a mesma linguagem), basta verificar se existe um isomorfismo entre $H/\approx^H_{Obs}$ e $H'/\approx^{H'}_{Obs}$.

$\diamond$

3.4 Refinamentos Comportamentais

Na Secção 2.4 foram apresentados os conceitos de refinamento entre especificações estruturadas, assim como o papel deste conceito no processo de desenvolvimento de software. A pertinência da adaptação destes conceitos à abordagem comportamental é evidente, na medida em que, segundo este prisma, a preservação dos requisitos por refinamento deixa de ser estrita para ser apenas comportamental. Assim, um refinamento é comportamentalmente correcto quando o seu comportamento preserva os requisitos da especificação refinada:

Definição 3.4.1 (Refinamento comportamental)**.** *Sejam SP e SP' especificações, $\approx$ uma Σ-igualdade comportamental isomorficamente compatível, e σ um morfismo de assinaturas. SP' é um σ-*refinamento comportamental *de SP a respeito de $\approx$ se*

$$\textbf{\textit{behaviour }} SP \textbf{\textit{ wrt }} \approx \ \rightsquigarrow_\sigma SP',$$

isto é, se

- $Sig(SP') = \sigma(Sig(\textbf{\textit{behaviour}}\ SP\ \textbf{\textit{wrt}}\ \approx))$ *e*
- $[\![SP']\!] \restriction_\sigma \subseteq [\![\textbf{\textit{behaviour}}\ SP\ \textbf{\textit{wrt}}\ \approx]\!]$,

$SP \rightsquigarrow^{\approx}_{\sigma} SP'$ *denota que* SP' *é* σ*-refinamento comportamental de* SP *segundo* $\approx$.

O recurso à utilização de refinamentos comportamentais na implementação de software é muito frequente, nomeadamente no caso observacional. Neste caso, os requisitos de SP só têm que ser satisfeitos observacionalmente, podendo não o ser estritamente (ver Exemplo 3.4.6). Os resultados e caracterizações que se obtêm do estudo dos refinamentos comportamentais (para igualdades comportamentais arbitrárias), são de grande importância no estudo do caso observacional feito na próxima Secção.

Na Secção 2.4, apresentou-se o método do refinamento passo-a-passo, como sendo um processo de refinamentos sucessivos, que a partir de uma especificação mais abstracta SP_0, produz uma outra especificação SP_n mais concreta, tal que $[\![SP_n]\!] \restriction_{\sigma'} \subseteq [\![SP_0]\!]$, para σ' composição dos morfismos dos passos de refinamento. A generalização deste método ao caso comportamental requer algum cuidado, na medida em que diferentes igualdades comportamentais podem ser consideradas nos vários passos de refinamento. A questão que se coloca é a de como controlar a conservação da propriedade da composicionalidade vertical neste processo. Alguns passos importantes neste estudo já estão dados como, por exemplo, o da caracterização de propriedades de suficiência para esta conservação. Para a apresentação destes resultados é necessária a introdução de alguns conceitos, nomeadamente ao nível da teoria das relações parciais.

Definição 3.4.2. *Sejam* A *uma* Σ*-álgebra e* θ_A *e* θ'_A *duas relações de congruência parcial em* A. *Diz-se que* θ_A é menor que θ'_A *sse:*

1. $Dom(\theta_A) \supseteq Dom(\theta'_A)$;
2. *Para quaisquer* $a, b \in Dom(\theta'_A)$, *se* $a\theta_A b$ *então* $a\theta'_A b$;

Escreve-se $\theta_A \leq \theta'_A$ *para denotar que* θ_A *é menor que* θ_A.

Dadas duas igualdades comportamentais $\approx$ e $\approx'$ em Σ, escreve-se $\approx \leq \approx'$ quando para toda a Σ-álgebra A se tem que $\approx^A \leq \approx'^A$. A importância deste conceito, quando se pretende trabalhar com refinamentos comportamentais, é bastante intuitiva. Por

exemplo, dadas duas igualdades comportamentais $\approx$ e $\approx'$ tais que $\approx' \leq \approx$, se uma especificação SP é fechada comportamentalmente por $\approx$, então

$$[\![\textbf{behaviour } SP \textbf{ wrt } \approx']\!] \subseteq [\![\textbf{behaviour } SP \textbf{ wrt } \approx]\!],$$

na medida em que a segunda relação distingue menos elementos do que a primeira (cf. [Hen97, Proposição 3.6.4]). Nestas condições, se $SP \rightsquigarrow^{\approx'} SP$ então $SP \rightsquigarrow^{\approx} SP$. Em particular, se $SP \rightsquigarrow SP'$, então $SP \rightsquigarrow^{\approx} SP'$ para qualquer $\approx$ tal que SP seja por si comportamentalmente fechada.

Definição 3.4.3 (Uniformidade de igualdades comportamentais)**.** *Seja Σ uma assinatura heterogénea. Uma Σ-igualdade comportamental $\approx = (\approx^A)_{A \in Alg(\Sigma)}$ diz-se* uniforme *se para qualquer Σ-igualdade $\approx' = (\approx'^A)_{A \in Alg(\Sigma)}$ tal que $\approx' \leq \approx$ e, para qualquer Σ-álgebra A, se tem que $A/\approx^A$ é isomorfo a $(A/\approx'^A)/\approx^{(A/\approx'^A)}$.*

São exemplos de igualdades comportamentais uniformes a igualdade observacional $\approx_{Obs}$ e a igualdade observacional parcial $\approx_{Obs,In}$. Estas provas, que são bastante extensas e tecnicamente complexas, podem ser consultadas em [Hen97, Apêndice A]. A proposição que se segue, tem um papel fulcral na demonstração do resultado central da Secção (Teorema 3.4.5). A sua demonstração pode ser encontrada, por exemplo, em [Hen97, Corolário 3.6.9].

Proposição 3.4.4. *Sejam SP e SP' especificações estruturadas, tais que $Sig(SP') = \sigma(Sig(SP))$, para σ morfismo de assinaturas. Sejam $\approx$ e $\approx'$ $Sig(SP)$-igualdade e $Sig(SP')$-igualdade comportamentais respectivamente, tais que $\approx$ seja uniforme e, para toda $A' \in Sig(SP')$-álgebra, se tem que $(\approx'^{A'}) \restriction_\sigma \leq \approx^{(A' \restriction_\sigma)}$.*

Então, se

$$[\![\ SP']\!] \restriction_\sigma \subseteq [\![\textbf{\textit{behaviour }} SP \textbf{\textit{ wrt }} \approx]\!]$$

tem-se que

$$[\![\textbf{\textit{behaviour }} SP' \textbf{\textit{ wrt }} \approx']\!] \restriction_\sigma \subseteq [\![\ \textbf{\textit{behaviour }} SP \textbf{\textit{ wrt }} \approx]\!].$$

*Ou seja, nestas condições, se $SP \rightsquigarrow^{\approx}_{\sigma} SP'$, então $SP \rightsquigarrow^{\approx}_{\sigma}$ **behaviour** SP' **wrt** $\approx'$.*

O Teorema que se segue, caracteriza as condições de suficiência, para que a composição de σ-refinamentos comportamentais, possua a propriedade da composicionalidade vertical. Este resultado é apresentado em [Hen97, Teorema 10.5.2] para o caso onde apenas se consideram morfismos de inclusão.

Teorema 3.4.5. *Sejam SP, SP' e SP'' especificações estruturadas com $Sig(SP') = \sigma(Sig(SP))$ e $Sig(SP'') = \tau(Sig(SP'))$. Sejam $\approx$ e $\approx'$ igualdades comportamentais em $Sig(SP)$ e $Sig(SP')$ respectivamente, tais que $\approx$ uniforme e que, para toda A' $Sig(SP')$-álgebra, $(\approx'^{A'}) \restriction_\sigma \leq \approx^{(A'\restriction_\sigma)}$. Se $SP \rightsquigarrow^{\approx}_{\sigma} SP'$ e $SP' \rightsquigarrow^{\approx'}_{\tau} SP''$, então $SP \rightsquigarrow^{\approx}_{\tau\circ\sigma} SP''$.*

Demonstração. Tem-se por hipótese que

$$[\![SP']\!] \restriction_\sigma \subseteq [\![\textbf{behaviour } SP \textbf{ wrt } \approx]\!] \tag{3.1}$$

e que

$$[\![SP'']\!] \restriction_\tau \subseteq]\!]\textbf{behaviour } SP' \textbf{ wrt } \approx' [\![. \tag{3.2}$$

Pela proposição anterior, tem-se que

$$[\![\textbf{behaviour } SP' \textbf{wrt } \approx']\!] \restriction_\sigma \subseteq [\![\textbf{behaviour } SP \textbf{ wrt } \approx]\!].$$

De (3.2) sai que

$$[\![SP'']\!] \restriction_\tau \restriction_\sigma \subseteq [\![\textbf{behaviour } SP' \textbf{ wrt } \approx']\!] \restriction_\sigma,$$

e portanto

$$[\![SP'']\!] \restriction_{\tau\circ\sigma} \subseteq [\![\textbf{behaviour } SP \textbf{ wrt } \approx]\!],$$

ou seja $SP \rightsquigarrow^{\approx}_{\tau\circ\sigma} SP''$. □

3.4.1 Refinamentos Observacionais

O conceitos de modelação observacional e o caso observacional do processo de refinamento passo-a-passo são de grande importância no processo de especificação e desenvolvimento formal de software[2]:

Exemplo 3.4.6 (STATE [HWB97])**.** *Considere-se a seguinte especificação:*

```
Spec STATEPARES = enrich ID by
```

[2]São exemplos clássicos deste tipo de refinamentos as implementações de *streams* (Exemplo 3.1.11) por um *array com ponteiro* (cf. [STar]) ou por um *array com contador* (cf. [Sch92]), implementação de *conjuntos* por *listas* (cf. [Hen97]), etc.;

```
[GEN]
    state;
    elt;
[CONS]
    init: -> state;
    < .,. >:id,elt -> state;
    _::_ :state,state -> state;
[OP]
    update:id,elt,state -> state;
    lookup:id,state -> elt;
[AX]
```

(∀s:state).s::init=s;

(∀s:state).init::s=s;

(∀s:state)(∀n:elm)(∀i:id).update(i,n,s)=<i,n>::s;

(∀i:id).lookup(i,init)=0;

(∀i:id)(∀n:elt)(∀s:state).lookup(i,<i,n>::s)=n;

(∀i,j:id)(∀s:state).i ≠ j → lookup(i,<j,d>::s)=lookup(i,s);

Observe-se que STATEPARES *não é um refinamento estrito da especificação* STATE *apresentada no Exemplo* 3.1.9*, uma vez que o requisito da comutatividade da operação* update *imposto em* STATE*, não é estritamente satisfeito por* STATEPARES*. No entanto, para Obs* = {id, elt} *o requisito é satisfeito observacionalmente, e portanto* STATE $\rightsquigarrow^{\approx_{Obs,S}}$ STATEPARES. ◇

A caracterização do processo de refinamento observacional passo-a-passo tem sido estudada por vários autores, sendo que, em todos estes trabalhos, se pressupõe a "preservação da observabilidade"das especificações entre passos de refinamento, no sentido em que, dadas duas especificações SP e SP' com conjuntos de géneros observáveis Obs e Obs', para que SP' seja σ-refinamento observacional SP, é imposto que se $s \in Obs$ então $\sigma(s) \in Obs'$ e que se $s \in S \setminus Obs$ então $\sigma(s) \in S' \setminus Obs'$ (cf.[GR99a, BHK03]), ou pelo menos que $\sigma(Obs) = Obs'$ (cf.[Mar06]). Contudo, a passagem de géneros não observáveis a observáveis e vice-versa, pode ser útil em diversas situações. Por exemplo, de acordo com o paradigma da orientação a objectos, apenas os dados de input/output

de um programa devem estar desencapsulados e, por razões de segurança, pode ser necessário encapsular alguns géneros de dados numa determinada fase do processo de refinamento. Por outro lado, desencapsular géneros durante este processo, pode ser vantajoso na aplicação de métodos de prova (por exemplo, caso se consiga desencapsular todos os géneros, tem-se que a relação $\approx_{Obs,In}$ pode ser interpretada pela relação $=$). Estuda-se nesta Secção o caso observacional do processo de refinamento passo-a-passo no qual não se impõe a preservação de observabilidade entre passos de refinamento, isto é, o processo de refinamento observacional no qual se admitem o encapsulamento e desencapsulamento de dados entre passos de refinamento. Este estudo é feito exclusivamente para o caso da igualdade observacional total e, em parte do trabalho, restringido a especificações equacionais. Esta imposição é importante na medida em que garante o fecho observacional de especificações por qualquer igualdade observacional (cf. Secção 3.3). Contudo, alguns destes resultados podem ser estendidos a outros tipos de especificações, pela imposição de algumas condições relativas ao seu fecho observacional. Os resultados que se seguem foram recentemente apresentados em [Mad08].

Considerem-se uma assinatura com conjunto de géneros observáveis Obs, e a mesma assinatura, com a passagem de alguns géneros de não observáveis a observáveis, isto é, com um novo conjunto de géneros observáveis Obs' tal que $Obs \subseteq Obs'$. Observe-se que a relação $\approx_{Obs'}$ é mais estrita do que $\approx_{Obs}$, na medida em $\mathcal{C}_\Sigma^{Obs} \subseteq \mathcal{C}_\Sigma^{Obs'}$, isto é, na medida em que na primeira são considerados mais contextos do que na segunda. Obviamente todos os modelos de uma especificação SP segundo $\models_{\approx_{Obs'}}$ também o são por $\models_{\approx_{Obs}}$ e, portanto,

$$[\![\textbf{behaviour } SP \textbf{ wrt } \approx_{Obs'}]\!] \subseteq [\![\textbf{behaviour } SP \textbf{ wrt } \approx_{Obs}]\!].$$

Pelas mesmas razões, dada uma assinatura Σ com um conjunto de géneros observáveis Obs_1, se se fizer o desencapsulamento progressivo de géneros da assinatura inicial, gera-se uma cadeira $Obs_1 \subseteq \cdots \subseteq Obs_n$ tal que $\approx_{Obs_1} \geq \approx_{Obs_2} \geq \cdots \geq \approx_{Obs_n}$. Chega-se assim a uma primeira caracterização da composicionalidade vertical de refinamentos observacionais: pelo Teorema 3.4.5, se se tiver $SP \rightsquigarrow^{\approx}_{Obs} SP'$ e $SP' \rightsquigarrow^{\approx}_{Obs'} SP''$ para $Obs \subseteq Obs'$, tem-se que $SP \rightsquigarrow^{\approx}_{Obs} SP''$. Estuda-se de seguida esta caracterização para o caso geral dos σ-refinamentos observacionais:

Definição 3.4.7 (Morfismo de preservação observacional fraca)**.** *Sejam $\Sigma = (S, \Omega)$ e $\Sigma' = (S', \Omega')$ duas assinaturas e Obs e Obs' conjuntos de géneros observáveis para Σ*

e Σ' *respectivamente. Um morfismo de assinaturas* $\sigma : \Sigma \to \Sigma'$ *diz-se* morfismo de $Obs - Obs'$-preservação observacional fraca *sse para todo* $s \in Obs, \sigma(s) \in Obs'$.

O resultado que se segue tem um papel fulcral na caracterização da composicionalidade vertical de refinamentos observacionais:

Teorema 3.4.8. *Para todo o morfismo de* $Obs - Obs'$*-preservação observacional fraca* $\sigma : \Sigma \to \Sigma'$*, e para toda a* Σ'*-álgebra* A'*, tem-se que*

$$(\approx^{A'}_{Obs'}) \restriction_\sigma \leq \approx^{(A'\restriction_\sigma)}_{Obs} .$$

Demonstração. Sejam $a, b \in A' \restriction_\sigma$ tais que $a(\approx^{A'}_{Obs'})\restriction_\sigma b$. Tem-se que $a(\approx^{A'}_{Obs'})b$, e portanto, para todos os contextos $c' \in \mathcal{C}^{Obs'}_{\Sigma'}$, para todas as valorações $\alpha'_1, \alpha'_2 : X' \cup \{Z_{\sigma(s)}\} \to A'$, tais que $\alpha'_1(x') = \alpha'_2(x')$ para todo o $x' \in X'$ e $\alpha'_1(z_{\sigma(s)}) = a$ e $\alpha'_2(z_{\sigma(s)}) = b$, tem-se que

$$I_{\alpha'_1}(c') = I_{\alpha'_2}(c'). \tag{3.3}$$

Uma vez que σ é morfismo de preservação contextual fraca, todos os contextos de $\mathcal{C}^{Obs}_{\Sigma}$ são mapeados por σ a contextos de $\mathcal{C}^{Obs'}_{\Sigma'}$. Assim, pelo facto de $A' \restriction_\sigma$ ser álgebra reduto de A', tem-se que todos os contextos considerados em $\approx^{A'\restriction_\sigma}_{Obs}$, também o são em $\approx^{A'}_{Obs'}$ (uma vez que por definição de álgebra reduto $c^{A'\restriction_\sigma} = \sigma(c)^{A'}$), e portanto de (3.3) tem-se em particular que, para todo o $c \in \mathcal{C}^{Obs}_{\Sigma}$, $I_{\alpha'_1}(c) = I_{\alpha'_2}(c)$.

Considerem-se agora as valorações reduto $\alpha'_1 \restriction_\sigma : X \to A' \restriction_\sigma$ e $\alpha'_2 \restriction_\sigma : X \to A' \restriction_\sigma$. Pelo Lema 2.2.3 tem-se que

$$\alpha'_1 \restriction_\sigma (x_s) = \alpha'_1(\sigma(x_s)) = \alpha'_1(x_{\sigma(s)}),$$

$$\alpha'_2 \restriction_\sigma (x_s) = \alpha'_2(\sigma(x_s)) = \alpha'_2(x_{\sigma(s)}),$$

$$\alpha'_1 \restriction_\sigma (z_s) = \alpha'_1(\sigma(z_s)) = \alpha'_1(z_{\sigma(s)}) \text{ e}$$

$$\alpha'_2 \restriction_\sigma (z_s) = \alpha'_2(\sigma(z_s)) = \alpha'_2(z_{\sigma(s)}).$$

Tem-se também que $\alpha'_1(z_{\sigma(s)}) = a$ e $\alpha'_2(z_{\sigma(s)}) = b$, e portanto, sai da unicidade de I, que para todo o contexto $c \in \mathcal{C}^{Obs}_{\Sigma}$, $I_{\alpha'_1\restriction_\sigma}(c) = I_{\alpha'_1}(c)$ e $I_{\alpha'_2\restriction_\sigma}(c) = I_{\alpha'_2}(c)$, ou seja, para todas as valorações $\alpha'_1, \alpha'_2 : X' \to A'$, para todo o contexto $c \in \mathcal{C}^{Obs}_{\Sigma}$,

$$I_{(\alpha'_1\restriction_\sigma)}(c) = I_{(\alpha'_2\restriction_\sigma)}(c). \tag{3.4}$$

Por outro lado, qualquer que seja a valoração $\alpha : X \to A' \restriction_\sigma$, existe uma valoração $\alpha' : X' \to A'$ tal que $\alpha = \alpha' \restriction_\sigma$ (todas as valorações $\alpha_s = \alpha'_{\sigma(s)}$), e portanto, sai de (3.4)

que para todas as valorações $\alpha_1, \alpha_2 : X' \to A' \restriction_\sigma$, tais que $\alpha_1(x') = \alpha_2(x')$, se $x' \in X'$, $\alpha_1(z_{\sigma(s)}) = a$, $\alpha_2(z_{\sigma(s)}) = b$, então, para todo o $c \in \mathcal{C}_\Sigma^{Obs}$, tem-se que $I_{\alpha_1}(c) = I_{\alpha_2}(c)$. Fica assim provado que $a(\approx_{Obs}^{A' \restriction_\sigma})b$.

□

Chega-se assim à seguinte caracterização da composição de refinamentos observacionais:

Corolário 3.4.9. *Sejam SP, SP' e SP'' especificações algébricas sobre as assinaturas Σ, Σ' e Σ'' com conjuntos de géneros observáveis Obs, Obs' e Obs'' respectivamente. Sejam σ um morfismo de $Obs - Obs'$-preservação observacional fraca e ϕ um morfismo de $Obs' - Obs''$-preservação observacional fraca. Então, se $SP \rightsquigarrow_\sigma^{\approx_{Obs}} SP'$ e $SP' \rightsquigarrow_\phi^{\approx_{Obs'}} SP''$, tem-se que $SP \rightsquigarrow_{\phi\circ\sigma}^{\approx_{Obs}} SP''$.*

Demonstração. A demonstração sai directamente dos Teoremas 3.4.5 e 3.4.8. □

Durante o processo de refinamento, pode ser necessário preservar a igualdade observacional no sentido de $(\approx_{Obs'}^{A'}) \restriction_\phi = \approx_{Obs}^{(A' \restriction_\phi)}$. A classe de morfismos que se define de seguida preserva isso mesmo e é utilizado na literatura na definição da *instituição da lógica observacional* (cf. [BHK03, Mar06]). O uso da terminologia *forte/fraca* foi feito com o intuito de distinguir a primeira caracterização desta mais restritiva:

Definição 3.4.10 (Morfismo de preservação observacional forte)**.** *Sejam $\Sigma = (S, \Omega)$ e $\Sigma' = (S', \Omega')$ duas assinaturas e Obs e Obs' conjuntos de géneros observáveis para Σ e Σ' respectivamente. Um morfismo de assinaturas $\sigma : \Sigma \to \Sigma'$ é um* morfismo de $Obs - Obs'$-preservação observacional forte *sse:*

- *Para todo $s \in Obs, \sigma(s) \in Obs'$;*
- *Para todo $s \in S \setminus Obs, \sigma(s) \in S' \setminus Obs'$;*
- *Para toda a operação $f' : s'_1, \ldots, s'_i, \ldots, s'_n \to s' \in \Sigma'$ tal que $s_i \in \sigma(S)$, existe uma função $f \in \Omega$ tal que $\phi(f) = f'$;*

Observe-se que este morfismo preserva a observabilidade dos géneros da assinatura Σ, e que qualquer que seja o Σ-contexto $c \in \mathcal{C}_\Sigma^{Obs}$, existe um Σ'-contexto $c' \in \mathcal{C}_{\Sigma'}^{Obs'}$ tal que $c' = \sigma(c)$, e vice-versa.

A introdução de todos os conceitos apresentados foi feita no sentido de encontrar condições suficientes para a preservação da propriedade da composicionalidade vertical

entre passos de refinamento. Contudo, esta composicionalidade só está assegurada quando processada pela relação de igualdade observacional considerada no primeiro passo de refinamento, e que seja tal que $\approx_{Obs_1} \geq \cdots \geq \approx_{Obs_n}$, (i.e. $Obs_1 \subseteq \cdots \subseteq Obs_n$). Assim, no processo

$$SP_0 \rightsquigarrow_{\sigma_1}^{\approx_{Obs_1}} SP_1 \rightsquigarrow_{\sigma_2}^{\approx_{Obs_2}} \cdots \rightsquigarrow_{\sigma_n}^{Obs_n} SP_n,$$

tal que $\approx_{Obs_1} \geq \cdots \geq \approx_{Obs_n}$,tem-se que $SP_0 \rightsquigarrow_{\sigma_n \circ \cdots \circ \sigma_1}^{\approx_{Obs_1}} SP_n$.

O estudo de outras caracterizações da composicionalidade vertical pode ser interessante, uma vez que, por exemplo, segundo a caracterização apresentada, todos os géneros desencapsulados durante os passos de refinamento tornam a ser encapsulados no final do processo. Pelas razões que se mencionaram no início da Secção, a possibilidade de compor verticalmente refinamentos observacionais, segundo a relação com mais géneros observáveis aparece muitas vezes como uma situação desejável. É esta a caracterização que se pretende fazer de seguida (para o caso das especificações equacionais).

Observe-se que para qualquer especificação equacional $SP = \langle \Sigma, \Phi \rangle$, e para qualquer $s \in Obs$, a relação $\approx_{Obs}$ é mais estrita do que a relação $\approx_{Obs\setminus\{s\}}$, isto é, $\approx_{Obs} \leq \approx_{Obs\setminus\{s\}}$. Assim, uma vez que $A \models_{\approx_{Obs}} \Phi \Rightarrow A \models_{\approx_{Obs\setminus\{s\}}} \Phi$, tem-se que

$$[\![\textbf{behaviour } SP \textbf{ wrt } \approx_{Obs}]\!] \subseteq [\![\textbf{behaviour } SP \textbf{ wrt } \approx_{Obs\setminus\{s\}}]\!].$$

Contudo, o reciproco não é verdadeiro. Para garantir esta reciprocidade é necessária a imposição de algumas condições sobre Φ. Por exemplo, quando $\Phi_s = \emptyset$ e $\Phi_{S\setminus Obs} = \emptyset$, tem-se que $A \models_{\approx_{Obs}} \Phi \Leftrightarrow A \models_{\approx_{Obs\setminus\{s\}}} \Phi$, e assim

$$[\![\textbf{behaviour } SP \textbf{ wrt } \approx_{Obs}]\!] = [\![\textbf{behaviour } SP \textbf{ wrt } \approx_{Obs\setminus\{s\}}]\!].$$

Nestas condições, podem-se compor os refinamentos observacionais $SP \rightsquigarrow_{\sigma}^{\approx_{Obs\setminus\{s\}}} SP'$ e $SP' \rightsquigarrow_{\tau}^{\approx_{Obs}} SP''$ pela relação $\approx_{Obs}$. Considere-se agora o seguinte resultado:

Lema 3.4.11. *Seja $\Sigma = (S, \Omega)$ uma assinatura, Obs um conjunto de géneros observáveis para Σ e Φ um conjunto de Σ-equações. Para qualquer Σ-álgebra A e para qualquer $s \in Obs$,*

$$A \models_{\approx_{Obs\cup\{v\}}} \Phi \textit{ sse } \left(A \models \Phi' \textit{ e } A \models_{\approx_{Obs}} \Phi\right)$$

onde

$$\Phi' = \Phi_v \cup \{c(t) = c(t') \mid t{=}t' \in \Phi_h, h \in S \setminus (Obs \cup \{v\}), c \in \mathcal{C}_{\Sigma}^{\{v\}}(h)\}.$$

Demonstração. Seja $A \models_{\approx_{Obs\cup\{v\}}} \Phi$. Uma vez que $A \models_{\approx_{Obs\cup\{v\}}} \Phi_{Obs\cup\{v\}}$ implica $A \models \Phi_{Obs\cup\{v\}}$, tem-se que

$$A \models_{\approx_{Obs}} \Phi_{Obs} \tag{3.5}$$

e

$$A \models \Phi_v. \tag{3.6}$$

Tem-se de (3.6) que $A \models_{\approx_{Obs}} \Phi_v$. Assim, uma vez que para todo o $h \in S\setminus(Obs\cup\{v\})$ $\mathcal{C}_\Sigma^{Obs}(h) \subseteq \mathcal{C}_\Sigma^{Obs\cup v}$ tem-se que $A \models_{\approx_{Obs}} \Phi_h$ para todo o $h \in S \setminus (Obs \cup \{v\})$. Portanto, $A \models_{\approx_{Obs}} \Phi$.

Seja agora $t{=}t' \in \Phi_h$, $h \in S \setminus (Obs \cup \{v\})$. Tem-se por hipótese que $A \models c(t){=}c(t')$ para qualquer $c \in \mathcal{C}_\Sigma^{Obs\cup v}$ (uma vez que $\mathcal{C}_\Sigma^{v} \subseteq \mathcal{C}_\Sigma^{Obs\cup\{v\}}$). Assim, sai de (3.6) que $A \models_{\approx_{Obs}} \Phi'$.

Suponha-se agora que $A \models_{\approx Obs\cup\{v\}} \Phi$ e $A \models \Phi'$. Seja $t{=}t \in \Phi_s$. Há três casos a considerar: (i) $s \in Obs$, (ii) $s = v$ and (iii) $s \in S \setminus Obs \cup \{v\}$. O primeiro caso é óbvio, uma vez que $A \models_{\approx_{Obs}} t{=}t'$ implica que $A \models t{=}t'$ e assim $A \models_{\approx_{Obs\cup\{v\}}} t{=}t'$. No caso (ii), tem-se que $A \models_{\approx_{Obs\cup\{v\}}} t{=}t'$ uma vez que por hipótese $A \models t{=}t'$. Para o caso (iii) basta observar que $\mathcal{C}_\Sigma^{Obs\cup\{v\}}(s) = \mathcal{C}_\Sigma^{Obs}(s) \cup \mathcal{C}_\Sigma^{\{v\}}(s)$. De $A \models \Phi'$ tem-se que $A \models c(t){=}c(t')$ para todo o $c \in \mathcal{C}_\Sigma^{\{v\}}(s)$, e de $A \models_{\approx_{Obs}} t{=}t'$ tem-se que $A \models c(t){=}c(t')$ para todo o $c \in \mathcal{C}_\Sigma^{Obs}(s)$. Assim $A \models_{\approx_{Obs\cup\{v\}}} t{=}t'$. □

Dada uma especificação equacional $SP = \langle \Sigma, \Phi \rangle$ define-se a especificação SP_v como sendo a especificação equacional $\langle \Sigma, \Phi' \rangle$ onde Φ' é o conjunto de equações definido no Lema 3.4.11. Observe-se que pelo Lema anterior se tem que

$$[\![\textbf{behaviour } SP \textbf{ wrt } \approx_{Obs\cup\{v\}}]\!] = [\![\textbf{behaviour } SP \textbf{ wrt } \approx_{Obs} + SP_v]\!].$$

O Teorema que se segue permite, a partir de um refinamento observacional a respeito de uma relação $\approx_{Obs}$ de uma especificação equacional, construir um refinamento observacional sobre a mesma especificação a respeito de uma relação mais estrita $\approx_{Obs\cup\{s\}}$. Este resultado pode ser útil, por exemplo, quando a determinada altura do processo de especificação, se decide desencapsular um determinado género da assinatura, sem desperdício do trabalho desenvolvido. Por outro lado, pode também ser útil quando, no decorrer do processo de especificação, ainda não estão exactamente decididos quais os géneros observáveis do sistema.

Teorema 3.4.12. *Sejam Φ uma conjunto de Σ-equações, $SP = \langle \Sigma, \Phi \rangle$ e SP' especificações e Obs um conjunto de géneros observáveis para Σ. Se $SP \rightsquigarrow^{\approx_{Obs}} SP'$ tem-se que*

$$SP \rightsquigarrow^{\approx_{Obs \cup \{s\}}} SP' + SP_s.$$

Demonstração. Tem-se por hipótese que $[\![SP]\!] \subseteq [\![\textbf{behaviour } SP \textbf{ wrt } \approx_{Obs}]\!]$ e assim $[\![SP]\!] \cap [\![SP_s]\!] \subseteq [\![\textbf{behaviour } SP \textbf{ wrt } \approx_{Obs}]\!] \cap [\![SP_s]\!]$. Pelo Teorema 3.4.11 tem-se que $[\![\textbf{behaviour } SP \textbf{ wrt } \approx_{Obs}]\!] \cap [\![SP_s]\!] = [\![\textbf{behaviour } SP \textbf{ wrt } \approx_{Obs \cup \{s\}}]\!]$ e portanto $SP \rightsquigarrow^{\approx_{Obs \cup \{s\}}} SP' + SP_s$. □

Exemplo 3.4.13. *Considere-se a especificação* `CELL1` *do Exemplo 2.4.3 e a seguinte especificação:*

`Spec CELL1`$_{cell}$`=`

```
[GEN]
    elt;
    cell;
[OP]
    put: elt,cell -> cell;
    get:cell -> elt;
[Ax]
(∀e,e':elt)(∀c:cell).put(e,put(e',c))) = put(e,c);
```

Tem-se pelo resultado anterior que a partir de um qualquer refinamento `CELL1` $\rightsquigarrow^{\approx_{\{elt\}}}$ *SP se pode construir o refinamento a respeito de uma relação de igualdade observacional mais estrita* `CELL1` $\rightsquigarrow^{\approx_{\{elt,cell\}}} SP +$ `CEll1`$_{cell}$.

Exemplo 3.4.14. *Considere-se agora o caso onde se associa à especificação* `CELLNAT` *(cf. Exemplo 2.4.6) uma álgebra booleana com um predicado* equality test[3] *associado ao género* `nat`*:*

```
Spec CELLNATBOOL = enrich CELLNAT by BOOL

Spec CELLNATEQ = enrich CELLNATBOOL by
[OP]
    eq:nat,nat ->bool;
[AX]
```

[3] *Um predicado* Equality test *é um predicado booleano que devolve o valor* true *se dois elementos forem iguais e que devolve* false *no caso contrário;*

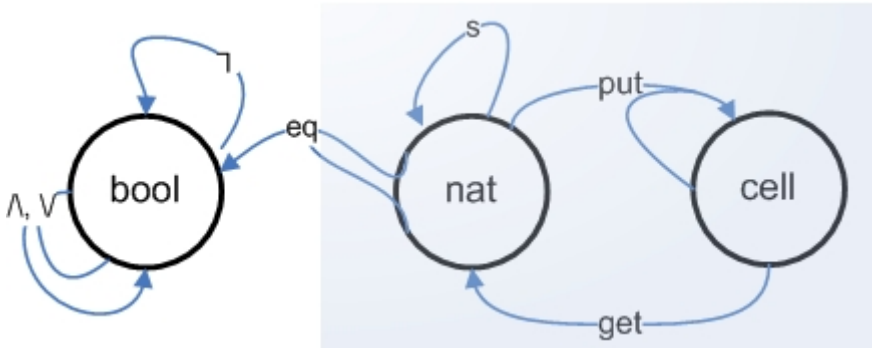

Figura 3.7: Diagrama da assinatura $\Sigma_{CELLNATEQ}$.

(∀x:nat).eq(x,x) = true;

(∀x,y:nat).eq(x,y) = true → eq(y,x) = true;

(∀x,y,z:nat).eq(x,y) = true ∧ eq(y,z) = true → eq(x,z) = true;

(∀x:nat).eq(s(p(x)),x) = true;

Suponha-se que se chegou a um refinamento CELLNATEQ $\rightsquigarrow^{\approx\{bool\}} SP$, *e que se decide, por alguma razão, desencapsular o género* nat. *Por forma a aproveitar o trabalho de especificação desenvolvido, define-se a especificação* $\texttt{CELLNATEQ}_{nat}$ *axiomatizada pelo conjunto de equações*

(∀x:nat).s(p(x)) = x;

(∀e,e':nat)(∀c:cell).get(put(e,put(e',c))) = get(put(e,c));

(∀e,e',e'':nat)(∀c:cell).get(put(e'',put(e,put(e'',c)))) =
get(put(e'',put(e,c)));

⋮

e chega-se ao refinamento observacional a respeito a {bool,nat} *por* SP+$\texttt{CELLNATEQ}_{nat}$.

Observe-se que o uso desta técnica tem que ser acompanhada de métodos complementares, na medida em que, no caso geral, o conjunto $\mathcal{C}_{\Sigma}^{Obs}(s)$ é infinito, induzindo portanto especificações SP_s infinitárias. Contudo, tal como se apresenta no próximo Capítulo, em grande parte dos casos é possível considerar apenas um conjunto finito de contextos $\mathcal{C}_{\Sigma} \subseteq \mathcal{C}_{\Sigma}^{\{s\}}$ em vez de todo o conjunto $\mathcal{C}_{\Sigma}^{\{s\}}$. Por outro lado, observe-se também que alguns dos resultados apresentados exclusivamente para especificações

equacionais, podem ser, sob algumas imposições, generalizados a outros tipos de especificações. Por exemplo, note-se que a definição do conjunto Φ' no Teorema 3.4.11, em nada depende das equações observáveis de Φ. Assim, pode-se estender este resultado, por exemplo, a todas as especificações axiomatizadas por um conjunto equações e equações condicionais com premissas observáveis.

3.5 Teoremas Comportamentais

Tal como se referiu na Secção 2.5, a verificação de propriedades é uma prática inerente a todo o processo de especificação algébrica de software sendo, portanto, necessário, o desenvolvimento de métodos de prova de teoremas comportamentais em especificações. O Teorema 3.1.4 estabelece uma relação directa entre a relação de satisfação comportamental e a relação de satisfação standard. No entanto, tal como foi observado, a aplicação prática deste resultado só é possível na verificação de propriedades em álgebras específicas. Falta responder à necessidade de verificar propriedades em especificações, isto é, em toda a classe dos seus modelos. Por forma a resolver este problema, *M. Bidoit* e *R. Hennicker* propõem em [BH96] um método, a partir do qual, a teoria comportamental de uma classe de álgebras é reduzida à teoria standard de uma outra, na qual se define a axiomatização da igualdade comportamental em questão. Esta nova classe de álgebras é obtida pela aplicação de um operador de álgebras e assinaturas denominado por *operador Lift*. Faz-se de seguida a apresentação deste método denominado por *método do operador Lift*.

3.5.1 O Método do Operador *Lift*

Dada uma especificação algébrica, o método de verificação comportamental que se apresenta é constituído por quatro etapas distintas: na primeira, enriquece-se a assinatura da especificação com um símbolo de predicado binário $\sim_s$ por cada género s da assinatura, por forma a denotar explicitamente a igualdade comportamental $\approx$ em s. Numa segunda etapa, denominada por *Axiomática Lifting*, faz-se uma axiomatização de $\approx$ à custa de fórmulas possivelmente infinitárias. Numa terceira e última etapa, a axiomatização infinitária construída na etapa anterior é transformada numa nova axiomatização finitária, feita à custa da introdução de novos géneros, novos símbolos de função e de novos axiomas. Esta axiomatização é denominada por *axiomatização*

finitária com parte escondida (*Hidden Part*).

A Secção está dividida em duas partes: na primeira, faz-se uma apresentação do método para o caso mais geral das igualdades comportamentais e, na segunda, para o caso mais específico da igualdade observacional.

O Caso Comportamental

Na primeira fase do método do operador *Lift*, por forma a denotar a igualdade observacional, a assinatura da especificação é enriquecida por um conjunto de símbolos de relação. As definições que se seguem são como que uma adaptação de alguns conceitos apresentados na Secção 2.1, adequados a este enriquecimento:

Definição 3.5.1 (Assinatura (Heterogénea) Relacional)**.** *Uma* assinatura relacional Σ^{rel} *é um par* (Σ, P)*, onde* $\Sigma = (S, \Omega)$ *é uma assinatura heterogénea e* P *uma* S^**-família de conjuntos de símbolos denominados por* símbolos de relação*.*

Note-se que esta definição pode ser vista como uma generalização do conceito de assinatura heterogénea (Definição 2.1.4), na medida em que qualquer assinatura Σ pode ser considerada como uma assinatura relacional com $P = \emptyset$. Por outro lado, observe-se que qualquer assinatura relacional pode ser vista como uma assinatura standard, se se tomar cada símbolo de relação $p \in P_{S^*}$ por um símbolo de função $p \in \Omega_{S^*,bool}$, isto é, por um predicado booleano.

Os conceitos apresentados na Secção 2.1 ajustam-se ao conceito de assinatura relacional de forma natural:

Definição 3.5.2 (Σ^{rel}-estrutura)**.** *Sejam* Σ *uma assinatura heterogénea e* $\Sigma^{rel} = (\Sigma, P)$ *uma assinatura relacional. Uma* Σ^{rel}-estrutura *é um triplo* $((A'_s)_{s\in S}, (f^{A'})_{f\in\Omega}, (p^{A'})_{p\in P})$*, onde* $((A'_s)_{s\in S}, (f^{A'})_{f\in\Omega})$ *é uma* Σ*-álgebra e todo* $p \in P_{s_1\cdots s_n}$ *é interpretado como uma relação* $p^{A'} \subseteq A'_{s_1} \times \cdots \times A'_{s_n}$*.*

Os conceitos de reduto, homomorfismo e congruência, são adatados ao caso relacional de forma natural. A relação *tarskiana* de satisfação entre Σ^{rel}-estruturas e Σ^{rel}-fórmulas estende-se da Definição 2.1.20 por:

Para toda a Σ^{rel}-estrutura A', para todos os termos $t_i \in T_{\Sigma'}(X)_{s_i}, 1 \leq i \leq n$ e para $\alpha : X \to A'$, $A', \alpha \models p(t_1, \ldots, t_n)$ se $(I_\alpha(t_1), \ldots, I_\alpha(t_n)) \in p^{A'}$.

A definição que se segue, descreve a forma pela qual uma assinatura Σ se transforma na assinatura $\mathcal{L}(\Sigma)$, assim como, o processo de transformação de uma Σ-fórmula ϕ numa $\mathcal{L}(\Sigma)$-fórmula $\mathcal{L}(\phi)$:

Definição 3.5.3 (*Lifting* de assinaturas e fórmulas)**.** *Seja* $\Sigma = (S, \Omega)$ *uma assinatura e seja* ϕ *uma* Σ*-fórmula.*

- $\mathcal{L}(\Sigma) =_{def} \Sigma \cup \{\sim_s\colon s, s | s \in S\}$.
- $\mathcal{L}(\phi) =_{def} \bigwedge_{y \in Var(\phi)} D_s(y) \Rightarrow \phi^*$*, onde* ϕ^* *se define da seguinte forma:*
  1. *Se* ϕ *do tipo* $l = r$ *com* $l, r \in s$*, então* $(l = r)^* =_{def} l \sim_s r$.
  2. $(\neg\phi)^* =_{def} \neg(\phi^*)$, $(\phi \wedge \psi)^* =_{def} \phi^* \wedge \psi^*$, $(\phi \vee \psi)^* =_{def} \phi^* \vee \psi^*$*, e de forma similar se definem conjunções e disjunções infinitárias.*
  3. $(\forall x{:}s.\phi)^* =_{def} \forall x{:}s.[D_s(x) \Rightarrow \phi^*]$.

onde $D_s(y)$ abrevia a expressão $y \sim_s y$.

Definição 3.5.4 (Semântica *Lifting* de uma álgebra)**.** *Seja* $\Sigma = (S, \Omega)$ *uma assinatura e seja* $\approx = (\approx^A)_{A \in Alg(\Sigma)}$ *uma igualdade comportamental.*

- *Para toda a* Σ*-álgebra* A*,* $\mathcal{L}(A)$ *é a (única) extensão de* A *a uma* $\mathcal{L}(\Sigma)$*-estrutura tal que:*
  1. $\mathcal{L}(A) \restriction_\Sigma = A$*;*
  2. *Para todo* $s \in S$*,* $\sim_s^{\mathcal{L}(A)} = \approx_s^A$.

Dada uma classe de Σ*-álgebras* C*,* $\mathcal{L}(C)$ *denota a classe* $\{\mathcal{L}(A) | A \in C\}$.

Pela aplicação deste operador, obtém-se este importante resultado:

Teorema 3.5.5. [BH96, Teorema 4.2.] *Sejam* $\approx$ *uma* Σ*-igualdade comportamental,* A Σ*-álgebra,* $\boldsymbol{C} \subseteq Alg(\Sigma)$ *classe de* Σ*-álgebras e* ϕ *uma* Σ*-fórmula. Tem-se que*

1. $A \models_\approx \phi$ *sse* $\mathcal{L}(A) \models \mathcal{L}(\phi)$*;*
2. $\boldsymbol{C} \models_\approx \phi$ *sse* $\mathcal{L}(\boldsymbol{C}) \models \mathcal{L}(\phi)$.

Nesta fase do método já está estabelecida a ponte entre as teorias comportamentais em C e as teorias standard na classe $\mathcal{L}(C)$. Perante este resultado, já se tem que dadas uma especificação estruturada SP e uma $Sig(SP)$-fórmula ϕ, $SP \models_\approx \phi$ sse $\mathcal{L}([\![SP]\!]) \models \mathcal{L}(\phi)$. Contudo, o trabalho de efectuar provas de teoremas comportamentais em $\mathcal{L}([\![SP]\!])$ não é geralmente possível, uma vez que apesar de $\approx$ estar representado pelos novos símbolos de predicado em $\mathcal{L}([\![SP]\!])$, ainda não está axiomatizada.

Definição 3.5.6. *Sejam $\Sigma = (S, \Omega)$ e Σ_1 duas assinaturas tais que $\Sigma \subseteq \Sigma_1$, e seja $\phi = (\phi_s(x_s, y_s))_{s \in S}$ uma família de Σ_1-fórmulas tais que, para todo o $s \in S$, $\phi_s(x_s, y_s)$ tem exactamente duas variáveis livres x_s e y_s, ambas do género s. Seja $Ax_\phi[\sim]$ a seguinte $\mathcal{L}(\Sigma) \cup \Sigma_1$-expressão:*

$$Ax_\phi[\sim] =_{def} \bigwedge_{s \in S} \forall x_s, y_s : s.[\phi_s(x_s, y_s) \Leftrightarrow x_s \sim_s y_s].$$

Para toda a Σ_1-álgebra A, $\mathcal{AL}_\phi(A)$ denota a (única) extensão de A a uma $\mathcal{L}(\Sigma) + \Sigma_1$-estrutura tal que:

1. *$\mathcal{AL}_\phi(A) \restriction_{\Sigma_1} = A$;*
2. *$\mathcal{AL}_\phi(A) \models Ax_\phi[\sim]$, ou seja, para todo $s \in S$, para quaisquer $a, b \in \mathcal{AL}_\phi(A)_s$, tem-se que $a \sim_s^{\mathcal{AL}_\phi(A)} b$ se e só se $\mathcal{AL}_\phi(A), \alpha \models \phi_s(x_s, y_s)$ para qualquer α tal que $\alpha(x_s) = a$ e $\alpha(y_s) = b$.*

Chama-se a $\mathcal{AL}_\phi(A)$ axiomática lifting *de A induzida pela S-família de fórmulas ϕ.*

O facto de se utilizar uma assinatura Σ_1 em vez da própria assinatura Σ na construção da *axiomática lifting*, deve-se a ser geralmente impossível fazer esta construção por Σ-fórmulas finitárias. Esta nova assinatura é obtida pela introdução de novos géneros e funções e é explicada mais à frente, na apresentação do conceito de axiomatização finitária com parte escondida.

Definição 3.5.7 (Axiomatização da igualdade comportamental $\approx$)**.** *Sejam $\Sigma = (S, \Omega)$ uma assinatura, $\approx$ uma Σ-igualdade comportamental e $Beh = (Beh_s(x_s, y_s))_{s \in S}$ uma família de Σ-fórmulas, tal que para todo o $s \in S$, $Beh_s(x_s, y_s)$ contenha apenas duas variáveis livres x_s e y_s, ambas do género s.*

A família Beh é uma axiomatização *para a Σ-igualdade $\approx$ se para toda a Σ-álgebra A, se tem que $\mathcal{AL}_{Beh}(A) = \mathcal{L}(A)$. Quando tal axiomatização existe, diz-se que $\approx$ é* axiomatizável.

A especificação *flat* $\langle \mathcal{L}(\Sigma), Ax_{Beh}[\sim] \rangle$, onde Beh é uma axiomatização de $\approx$ e $Ax_{Beh}[\sim]$ definida como em 3.5.6, designa-se por *especificação da igualdade comportamental* $\approx$ induzida por Beh. Observe-se que, dada uma axiomatização Beh de uma Σ-igualdade comportamental $\approx$ e uma classe de Σ-álgebras C, tem-se que $\mathcal{L}(C) = C + \langle \mathcal{L}(\Sigma), Ax_{Beh}[\sim] \rangle$, uma vez que por definição, $\mathcal{L}(C) = \{\mathcal{L}(A) | A \in C\}$, e porque Beh axiomatização de $\approx$, tem-se que $\mathcal{L}(C) = \{\mathcal{AL}_{Beh}(A) | A \in C\} = \{B \in Alg(\mathcal{L}(\Sigma)) | B \restriction_\Sigma \in C$ e $B \models AX_{Beh}[\sim]\}$, isto é, $C + \langle \mathcal{L}(\Sigma), Ax_{Beh}[\sim] \rangle$.

Exemplo 3.5.8 (A axiomatização (infinitária) de $\approx_{Obs,In}$)**.** *Sejam $\Sigma = (S, \Omega)$ uma assinatura, $Obs \subseteq S$ conjunto de géneros observáveis e $In \subseteq S$ conjunto de géneros de input sensível a Σ. A Σ-igualdade observacional $\approx_{Obs,In}$ induzida por Obs e In é axiomatizada pela S-família de Σ-fórmulas $Beh = (Beh_s(x_s, y_s))_{s \in S}$, definida para cada $s \in S$ da seguinte forma:*

$$Beh_s(x_s, y_s) = D_s(x_s) \wedge D_s(y_s) \wedge \bigwedge_{c[z_s] \in \mathcal{C}_\Sigma^{Obs}} \forall Var(c).D(Var(c)) \Rightarrow c[x_s] = c[z_s],$$

onde $D_s(x)$ abrevia a expressão $\bigvee_{t \in T_\Sigma(X_{In})_s} \exists var(t).x = t$, e $D(Var(t))$ abrevia $\bigwedge_{t_{s'} \in Var(t), s' \in S/In} D_{s'}(t_{s'})$. Observe-se que estas expressões são infinitárias, na medida em que os conjuntos $\mathcal{C}_\Sigma^{Obs}$ e $T_\Sigma(X_{In})_s$ são geralmente infinitos, gerando por isso conjunções e disjunções infinitárias.

◇

Teorema 3.5.9. [BH96, Teorema 5.5] *Sejam $\approx$ uma Σ-igualdade comportamental, Beh uma axiomatização para $\approx$, A uma Σ-álgebra, $\boldsymbol{C} \subseteq Alg(\Sigma)$ uma classe de Σ-álgebras e ϕ uma Σ-fórmula. Então, tem-se que:*

1. *$A \models_\approx \phi$ sse $\mathcal{AL}_{Beh}(A) \models \mathcal{L}(\phi)$;*
2. *$\boldsymbol{C} \models_\approx \phi$ sse $\boldsymbol{C} + \langle \mathcal{L}(\Sigma), Ax_{Beh}[\sim] \rangle \models \mathcal{L}(\phi)$.*

Nesta fase do método, já se consegue caracterizar a teoria comportamental de uma classe C pela teoria standard $\mathcal{L}(C)$, onde $\approx$ está axiomatizada. Contudo, tal como se pode ver no Exemplo 3.5.8, estas axiomatizações são geralmente infinitárias e, por conseguinte, a aplicação de métodos de prova continua a ser difícil (ou mesmo impossível)[4]. Por forma a resolver esta limitação, o método do operador *Lift* transforma as axiomatizações infinitárias em axiomatizações finitárias, a partir da introdução de novos géneros e novos símbolos de função. O objectivo deste enriquecimento é o de obter uma linguagem com maior poder de expressão, por forma a conseguir expressar o mesmo, mas agora sob a forma de expressões finitárias. Outra consideração que se deve ter é que a nova noção de axiomatização, contrariamente à da Definição 3.5.6, não é construída para toda a Σ-álgebra $A \in Alg(\Sigma)$, mas apenas para uma classe $C \subseteq Alg(\Sigma)$, o que não limita os objectivos da mesma, uma vez que quando se pretende verificar uma propriedade (comportamental) ϕ numa especificação SP, ela só tem que ser verificada na classe de modelos $[\![SP]\!] \subseteq Alg(\Sigma)$.

[4] Já em [Wol87], e posteriormente em [Sch92], se mostra a inexistência de axiomatizações finitárias para a igualdade observacional em algumas lógicas observacionais, nomeadamente, a do Exemplo 3.1.11. Este assunto é abordado no Capítulo 5;

Definição 3.5.10 (Axiomatização finitária de $\approx$ com *Parte escondida*)**.** *Sejam* $\Sigma = (S, \Omega)$ *uma assinatura,* $\approx$ *uma* Σ*-igualdade comportamental e* $C \subseteq Alg(\Sigma)$ *uma classe de* Σ*-álgebras. Seja* HID = ***reach*** $\langle \Sigma_H, \Phi_H \rangle$ ***with*** $\Omega_{\mathcal{R}}$ *uma especificação* flat *com* reachability constraint *(Definição 2.3.5) tal que* $\Sigma \subseteq \Sigma_H$*, e seja* $\mathrm{HBeh} = (\mathrm{HBeh}_s(x_s, y_s)_{s \in S})$ *uma* S*-família de* Σ_H*-fórmulas finitárias, tais que* $\mathrm{HBeh}_s(x_s, y_s)$ *contenha exactamente duas variáveis livres* x_s *e* y_s*, ambas do género* s*. O par* (HID,HBeh) *é uma* axiomatização finitária com Parte Escondida *da* Σ*-igualdade comportamental* $\approx$ *na classe de* Σ*-álgebras* C *se as seguintes condições se verificam:*

1. $(C + \mathrm{HID}) \restriction_{\Sigma} = C$*, ou seja, para toda a* Σ*-álgebra* $A \in C$ *existe pelo menos uma* $A_H \restriction_{\Sigma} = A$ *tal que* $A_H \in C + \mathrm{HID}$*;*

2. *Para qualquer* $A \in C$ *e para qualquer* $A_H \in [\![\mathrm{HID} + \mathrm{C}]\!]$*, tais que* $A_H \restriction_{\Sigma} = A$*, tem-se que* $\mathcal{AL}_{\mathrm{HBeh}}(A_H) \restriction_{\mathcal{L}(\Sigma)} = \mathcal{L}(A)$*.*

Ajusta-se agora o conceito de axiomatização de $\approx$:

Definição 3.5.11 (Especificação finitária com parte escondida)**.** *Sejam* $\approx$ *uma* Σ*-igualdade comportamental e* (HID,HBeh) *uma axiomatização finitária com parte escondida* HID *de* $\approx$ *a respeito a* $C \subseteq Alg(\Sigma)$*. Seja*

$$FinAx_{\mathrm{HBeh}}[\sim] = \bigwedge_{s \in S} \forall x_s, y_s : s.[\mathrm{HBeh}_s(x_s, y_s) \Leftrightarrow x_s \sim_s y_s].$$

A especificação $\mathrm{HID} + \langle \mathcal{L}(C) + \Sigma_H, FinAx_{\mathrm{HBeh}}[\sim] \rangle$ *é denominada por* especificação finitária com parte escondida HID *de* $\approx$ *em* C*.*

Teorema 3.5.12. [BH96, Teorema 6.3] *Sejam* $\approx$ *uma* Σ*-igualdade comportamental e* $C \subseteq Alg(\Sigma)$ *uma classe de* Σ*-álgebras. Seja* (HID, HBeh) *uma axiomatização com parte escondida de* $\approx$ *em* C*.*

1. *Para toda* Σ*-álgebra* A *em* C *e* $A_H \in [\![C + \mathrm{HID}]\!]$ *tal que* $A_H \restriction_{\Sigma} = A$*, tem-se que* $A \models_{\approx} \phi$ *sse* $\mathcal{AL}_{\mathrm{HBeh}}(A_H) \models \mathcal{L}(\phi)$*;*

2. $C \models_{\approx} \phi$ *sse* $C + \mathrm{HID} + \langle \mathcal{L}(\Sigma) + \Sigma_H, FinAx_{\mathrm{HBeh}}[\sim] \rangle \models \mathcal{L}(\phi)$*.*

O Caso Observacional

Na Secção anterior, descreveu-se um método a partir do qual, a teoria comportamental de uma classe de álgebras é reduzida à teoria standard de uma outra classe, para

a qual está definida uma axiomatização finitária da igualdade comportamental em questão. Nesta Secção, particulariza-se este resultado ao caso da igualdade observacional total, sendo que o caso parcial também é tratado, por exemplo, em [BH96]. Em [BH96], é descrito um algoritmo para a construção de axiomatizações finitárias para qualquer igualdade observacional parcial e qualquer classe de álgebras, a partir da codificação dos contextos e dos termos alcançáveis por input na parte escondida. A aplicação destes contextos e a interpretação dos termos é aqui especificada por símbolos de função da parte escondida. Uma adaptação deste trabalho à terminologia da lógica Γ-observacional é posteriormente apresentada por *Goguen* e *Roşu* em [GR99a] (denominado por processo de *Unhiding*). Segundo os próprios autores, o interesse destes métodos é estritamente teórico, uma vez que o processo da codificação dos contextos é tão complexo como o processo de indução sobre os mesmos. No entanto, tal como se mostra nos exemplos desta Secção, a aplicação dos resultados da Secção anterior pode ser útil na prática da verificação comportamental de propriedades, quando feita *ad hoc*. No final da Secção apresenta-se um algoritmo para a construção de axiomatizações finitárias para qualquer igualdade observacional (total) em qualquer especificação.

A Axiomatização Finitária da Σ-Igualdade Observacional

Tal como se pôde observar no Exemplo 3.5.8, a axiomatização da igualdade observacional é em geral infinitária, devido à natureza dos conjuntos de contextos observáveis $\mathcal{C}_\Sigma^{Obs}$, e dos termos construídos por variáveis de input $T_\Sigma(X_{In})$. Estuda-se, de seguida, o processo pelo qual se ultrapassa a primeira causa de infinitaridade. Seja $Beh = (Beh_s(x_s, y_s))_{s\in S}$ uma axiomatização (infinitária) da igualdade observacional total, definida para cada género $s \in S$ da seguinte forma:

$$Beh_s(x_s, y_s) = \bigwedge_{c\in\mathcal{C}_\Sigma^{Obs}} \forall Var(c).c[x_s] = c[y_s].$$

O método do operador *Lift* contorna a questão da conjunção infinitária de contextos da seguinte forma:

1. Procura um conjunto de contextos finito $\mathcal{C}$ tal que $\approx_{\mathcal{C}}^{A} \subseteq \approx_{Obs}^{A}$ (basta que para todo o $s \in Obs$, $z_s \in \mathcal{C}$ e que $\approx_{\mathcal{C}}^{A}$ seja congruência - Teorema 3.1.22).

2. Caso não seja possível, completa a parte escondida da especificação pela introdução de símbolos de função, a partir dos quais seja possível construir um conjunto de contextos $\mathcal{C}_H$ apropriado.

No exemplo que se segue, é possível encontrar o conjunto de contextos $\mathcal{C}$ referido em 1.:

Exemplo 3.5.13 (FLAGS)**.** *Considere-se a especificação* FLAGS *do Exemplo 3.1.10. Pretende-se provar que* $\texttt{FLAGS} \models_{\approx_{Obs}} \texttt{rev(rev(x))=x}$ *(observe-se que a propriedade não é um teorema standard).*[5] *Seja* $A \in [\![\texttt{FLAGS}]\!]$ *arbitrária. Uma vez que o conjunto de contextos* $\mathcal{C}^{Obs}_{\Sigma_{\texttt{FLAGS}}}$ *é infinito (Exemplo 3.1.14), tem-se que a axiomatização de* $\approx_{Obs}$ *é infinitária. Considere-se o conjunto de contextos* $\mathcal{C} = \{\texttt{z}_{bool}, \texttt{?up(z}_{flag}\texttt{)}\}$*. Uma vez que* $\texttt{z}_{bool} \in \mathcal{C}$*, basta que* $\approx^A_{\mathcal{C}}$ *seja congruência parcial, para que* $\approx^A_{Obs} = \approx^A_{\mathcal{C}}$ *(Teorema 3.1.22). Pela Definição 3.1.15, tem-se que*

$$x_{flag} \approx^A_{\mathcal{C}} y_{flag} \text{ sse } \texttt{?up(x}_{flag}\texttt{)} = \texttt{?up(y}_{flag}\texttt{)} \text{ e}$$

$$\texttt{x}_{bool} \approx^A_{\mathcal{C}} \texttt{y}_{bool} \text{ sse } \texttt{x}_{bool} = \texttt{y}_{bool}.$$

Tem-se que se $\texttt{x}_{flag} \approx^A_{\mathcal{C}} \texttt{y}_{flag}$*, então* $\texttt{up(x}_{flag}\texttt{)} \approx^A_{\mathcal{C}} \texttt{up(y}_{flag}\texttt{)}$ *e* $\texttt{down(x}_{flag}\texttt{)} \approx^A_{\mathcal{C}} \texttt{down(y}_{flag}\texttt{)}$ *uma vez que* $(\forall \texttt{x} : \texttt{flag})\texttt{?up(up(x))=true}$ *e* $(\forall \texttt{x} : \texttt{flag})\texttt{?up(down(x))=false}$*. Por outro lado, uma vez que* $(\forall \texttt{x} : \texttt{flag})\texttt{?up(rev(x))=not(?up(x))}$*, tem-se que*

$$\texttt{?up(rev(x}_{flag}\texttt{))=?up(rev(y}_{flag}\texttt{))}$$

é o mesmo que $\texttt{not(?up(x}_{flag}\texttt{))=not(?up(y}_{flag}\texttt{))}$*, o que se verifica, uma vez que* $\texttt{?up(x}_{flag}\texttt{)} = \texttt{?up(y}_{flag}\texttt{)}$*. Prova-se desta forma que* $\approx^A_{\mathcal{C}}$ *é relação de congruência escondida, e portanto,* $\approx^A_{Obs} = \approx^A_{\mathcal{C}}$*. Assim, a conjunção infinita de contextos em Beh passa a finita pela substituição de* $\mathcal{C}^{Obs}_{\Sigma_{\texttt{FLAGS}}}$ *por* $\{\texttt{?up(z}_{flag}\texttt{)},\texttt{z}_{bool}\}$*.*

Agora,

$$\texttt{FLAGS} \models_{\approx_{Obs}} \texttt{rev(rev(x))=x} \text{ sse } \texttt{FLAGS}{+}\langle \mathcal{L}(\texttt{FLAGS}), Ax_{Beh}[\sim]\rangle \models_{\approx^A_{\mathcal{C}}} \texttt{rev(rev(x))} \sim_{flag} \texttt{x},$$

onde $Ax_{Beh}[\sim] = \bigwedge_{s\in S} Ax_{Beh}[\sim_s]$ *é definido por*

$$Ax_{Beh}[\sim_{bool}] = \forall \texttt{x, y:bool}.[\texttt{x} \sim_{bool} \texttt{y} \Leftrightarrow \texttt{x=y}], \text{ e}$$

$$Ax_{Beh}[\sim_{flag}] = \forall \texttt{x, y:flag}.[\texttt{x} \sim_{flag} \texttt{y} \Leftrightarrow \texttt{?up(x)=?up(y)}].$$

De facto, $\texttt{?up(rev(rev(x)))=not(?up(rev(x)))=not(not(?up(x)))=?up(x)}$*, e portanto* $\texttt{FLAGS} + \langle \mathcal{L}(\texttt{FLAGS}), Ax_{Beh}[\sim]\rangle \models_{\approx^A_{\mathcal{C}}} \texttt{rev(rev(x))} \sim_{flag} \texttt{x}$*, tal como se pretendia.*

◇

[5] *Em* [Mar04, Exemplo 1.2.3.3] *pode-se encontrar um exemplo de um modelo observacional desta especificação que não satisfaz estritamente esta propriedade.*

Tal como já foi referido, contrariamente ao que aconteceu no exemplo anterior, muitas vezes é impossível encontrar conjuntos finitos de contextos $\mathcal{C}$ substituíveis pelo conjunto de contextos $\mathcal{C}_\Sigma^{Obs}$. Quando tal acontece, procede-se à introdução de uma parte escondida na especificação SP (ver Secção 3.5.1). O critério que se segue caracteriza a parte escondida para a axiomatização da igualdade observacional total. Observe-se que esta caracterização generaliza também o caso da axiomatização sem parte escondida, uma vez que esta pode ser vista como o caso onde HID= $\langle \emptyset, \emptyset \rangle$. Segue-se a caracterização das axiomatizações finitárias de $\approx_{Obs}$:

Teorema 3.5.14. [BH96, Teorema 7.11] *Seja C uma classe de Σ-álgebras. Seja* HID = ***reach*** $\langle \Sigma_H, \mathcal{A}x_H \rangle$ ***by*** Ω_H, *com $\Sigma \subseteq \Sigma_H$. Sejam $\mathcal{C}_H$ um conjunto arbitrário de Σ_H-contextos, e* HBeh$[\mathcal{C}_\mathcal{H}]$ *o seguinte S-conjunto de Σ_H-fórmulas definidas para cada s da seguinte forma:*

$$\mathrm{HBeh}_s[\mathcal{C}_\mathcal{H}](x_s, y_s) = \bigwedge_{c_H \in \mathcal{C}_\mathcal{H}(s)} \forall Var(c_H) c_H[x_s] = c_H[y_s].$$

Então, (HID,HBeh$[\mathcal{C}_\mathcal{H}]$) *é uma axiomatização finitária com parte escondida da igualdade comportamental $\approx_{Obs}$ a respeito de C se:*

1. $(C + \mathrm{HID}) \restriction_\Sigma = \boldsymbol{C}$;
2. *Para qualquer Σ_H-álgebra $A_H \in [\![C + \mathrm{HID}]\!]$, $\mathcal{AL}_{\mathrm{HBeh}}(A_H) \models Cong_\Sigma^\sim$, isto é, $(C + \mathrm{HID} + \langle \Sigma_H^\mathcal{L}, Ax_{\mathrm{HBeh}[\mathcal{C}_H]} \rangle) \models Cong_\Sigma^\sim$, onde $Cong_\Sigma^\sim = \bigwedge_{f \in F} Cong_f^\sim$, onde para cada $f \in \Omega_{s_1 \cdots s_n, s}$,*

   $$Cong_f^\sim = \forall x_1, y_1 : s_1, \ldots, x_n, y_n : s_n.[(\bigwedge_{1 \leq i \leq n} x_i \sim_{s_i} y_i) \Rightarrow f(x_1, \ldots, x_n) \sim_s f(y_1, \ldots, y_n)];$$

3. *existe um conjunto (possivelmente infinito) $\mathcal{C} \subseteq \mathcal{C}_\Sigma^{Obs}$ tal que para todo o $s \in Obs$, $z_s \in \mathcal{C}$, e tal que satisfaça as seguintes condições:*

   - *Para cada $s \in S$, para cada $c_H \in \mathcal{C}_H$, e para cada $\alpha : X_H \to A_H$, existe um contexto $c \in \mathcal{C}(s)$, e um $\beta : X \to A_H$ tais que para qualquer $a \in (A_H)_s$ se tem $I_{\beta_a}(c) = I_{\alpha_a}(c_H)$, onde I_{α_a} e I_{β_a} são as únicas extensões de I_α e de I_β definidas por $I_{\alpha_a}(z_s) = I_{\beta_a}(z_s) = a$;*
   - *Para cada $s \in S$, para cada $c \in \mathcal{C}(s)$, e para cada $\alpha : X \to A_H$, existe um contexto $c_H \in \mathcal{C}_H$, e um $\beta : X_H \to A_H$ tais que para qualquer $a \in (A_H)_s$ se tem $I_{\beta_a}(c_H) = I_{\alpha_a}(c)$, onde I_{α_a} e I_{β_a} são as únicas extensões de I_α e de I_β definidas por $I_{\alpha_a}(z_s) = I_{\beta_a}(z_s) = a$.*

Apresenta-se de seguida a axiomatização finitária com parte escondida da igualdade observacional da especificação STREAM do Exemplo 3.1.11. Esta escolha é interessante, na medida em que em [Wol87][6] é demonstrada a impossibilidade de especificar esta igualdade a partir de um conjunto finito de axiomas:

Exemplo 3.5.15 (STREAM (Adaptado de [BH96]))**.** *Considere-se a especificação* STREAM *do Exemplo 3.1.11. O conjunto dos contextos cruciais de* Σ_{STREAM} *é o conjunto* $\{\mathtt{z}_{elt}, \mathtt{top}(\mathtt{z}_{stream})\}$*. Obviamente, este conjunto não é suficiente para axiomatizar a igualdade observacional* $\approx^{A}_{Obs}$*, uma vez que só considera os elementos do topo das* streams *e intuitivamente se compreende que duas* streams *são observacionalmente iguais, quando possuem os mesmos elementos* elt*, segundo a mesma ordem. Por esta razão, o conjunto de contextos a considerar teria de ser* $\{\mathtt{head}(\mathtt{tail}^n(\mathtt{z}_{stream}))|n \in \mathbb{N}\} + \{\mathtt{z}_{elt}\}$*. Considere-se a seguinte especificação* HID*:*

```
Spec HID = enrich NAT + STREAM by
[OP]
    headn: nat,stream -> elt;
[AX]
```

(∀n:nat,l:stream).headn(0,l))=head(l)*;*

(∀x:elt,l:stream).headn(s(n),l)=headn(n,pop(l))*;*

onde NAT *representa a especificação clássica dos números naturais. Agora, pelo uso do novo símbolo de função* headn(n,s)*, já é possível definir um conjunto finito de contextos para axiomatizar finitamente a igualdade observacional, nomeadamente o conjunto* $\mathcal{C}_H = \{\mathtt{headn}(\mathtt{n},\mathtt{z}_{stream}), \mathtt{z}_{elt}\}$ *e a axiomatização finitária com parte escondida:*

$$\mathrm{HBeh}[\mathcal{C}_H]_{Stream}(\mathtt{sL},\mathtt{sR}) = \forall \mathtt{n{:}nat}\,.\,[\mathtt{headn(n,sL)=headn(n,sR)}],$$

$$\mathrm{HBeh}[\mathcal{C}_H]_{elt}(\mathtt{x}_{elt},\mathtt{y}_{elt}) = [\mathtt{x}_{elt} = \mathtt{y}_{elt}].$$

Tem-se que

$$\mathcal{L}(Sig(\mathtt{STREAM})) = Sig(\mathtt{STREAM}) \cup \{\sim_{stream}\colon \mathtt{stream} \times \mathtt{stream}\},$$

$FinAx_{\mathrm{HBeh}[\mathcal{C}_H]} = \forall \mathtt{sL},\mathtt{sR{:}stream}.[\mathtt{sL} \sim_{stream} \mathtt{sR} \Leftrightarrow (\forall \mathtt{n{:}nat}\,.\,\mathtt{headn(n,sL)=topn(n,sR)})]$*, e* $Cong_{Sig(\mathtt{STREAM})}[\sim] = \forall \mathtt{sL},\mathtt{sR{:}stream}, \forall \mathtt{x{:}elt}\,.\ [(\mathtt{sL} \sim_{stream} \mathtt{sR}) \Rightarrow \mathtt{x::sL} \sim_{stream} \mathtt{x::sR} \wedge \mathtt{tail(sL)} \sim_{stream} \mathtt{tail(sR)} \wedge \mathtt{head(sL)} \sim_{stream} \mathtt{head(sR)}]$.

[6] Neste trabalho, a especificação *STREAM* aparece denominada por especificação *STACK*;

Observe-se que o ponto 3. do critério anterior verifica-se sempre, uma vez que para toda a valoração $\alpha : X \to \mathbb{N}$*, o* $\mathcal{C}_H$*-contexto* $\mathtt{headn(x,z}_{stream})$ *corresponde ao* $\mathcal{C}^{Obs}_{\mathtt{STREAM}}$*-contexto* $\mathtt{head(tail}^{\alpha(x)}\mathtt{(z}_{stream}))$*, e vice versa.*

Assim, dada uma $\Sigma_{\mathtt{STREAM}}$*-álgebra* A *tal que* HID *seja extensão conservativa de* A*, e que*

$$A + \mathrm{HID} + \langle \mathcal{L}(Sig(\mathtt{STREAM})), FinAx_{\mathrm{HBeh}[\mathcal{C}_H]} \rangle \models Cong_{Sig(\mathtt{STREAM})[\sim]}, \text{ tem-se que}$$

$$A \models_{\approx_{Obs}} \phi \text{ sse } A + \mathrm{HID} + \langle \mathcal{L}(Sig(\mathtt{STREAM})), FinAx_{\mathrm{HBeh}[\mathcal{C}_H]} \rangle \models \mathcal{L}(\phi),$$

e portanto, para verificar se A *realiza correctamente* STREAM*, basta verificar se para todos os axiomas* ϕ *de* STREAM*,* $A + \mathrm{HID} + \langle \mathcal{L}(Sig(\mathtt{STREAM})), FinAx_{\mathrm{HBeh}[\mathcal{C}_H]} \rangle \models \mathcal{L}(\phi)$*.*

◇

Construção de Axiomatizações Finitárias para Igualdades Observacionais Arbitrárias

Tal como foi referido na Secção 3.5.1, apresenta-se em [BH96] um método a partir do qual se constroem axiomatizações finitárias para qualquer igualdade observacional parcial em qualquer classe de álgebras.

Apresenta-se nesta Secção a particularização deste método ao caso total. Considere-se a seguinte definição:

Definição 3.5.16 (Codificação da definição de contextos)**.** *Sejam* $\Sigma = (S, \Omega)$ *assinatura e* $Obs \subseteq S$ *conjunto dos géneros observáveis de* Σ*. Seja* $\mathrm{HID}[\Sigma, Obs] =$ ***reach*** $\langle \Sigma_H, \Phi_H \rangle$ ***by*** $\mathcal{R}_H$*, com* $\Sigma_H = (S_H, \Omega_H)$ *a especificação definida da seguinte forma:*

1. *Seja* $Ar[\Sigma] = \{s' \to s | s, s' \in S,$
   e existe pelo menos um contexto do género s *com variável contextual* $s'\}$*;*
2. $S_H = S \cup \{s_{[s' \to s]} | s' \to s \in Ar[\Sigma]\}$*;*
3. *Define-se* Ω_H *como sendo o menor conjunto de símbolos de função tal que:*
   (a) *Para cada* $s \in S$*, uma constante* $k_s \in \Omega_{H\, s[s \to s]}$*;*
   (b) *Para cada* $f : s_1, \ldots, s_n \to s \in \Sigma$*, para cada* $i \in \{1, \ldots, n\}$ *e, para cada* s' *tal que* $s' \to s_i \in Ar[\Sigma]$*,* $f_{i,s'} : s_1, \ldots, s_{[s' \to s_i]}, \ldots, s_n \to s_{s' \to s} \in \Sigma_H$*;*

(c) *Para cada* $s' \to s \in Ar[\Sigma]$, $apply_{s'\to s} \in \Omega_{H\, s_{[s'\to s]}s',s}$;

4. $\mathcal{R}_H = (S_\mathcal{R}, \Omega_\mathcal{R})$, *onde*

   (a) $S_\mathcal{R} = S_H \setminus S$;

   (b) $\Omega_\mathcal{R} = \{k_s | s \in S\} \cup \{f_{i,s'} | f : s_1, \ldots, s_n \to s \in \Sigma, 1 \leq i \leq n \text{ e } s' \to s_i \in Ar[\Sigma]\}$;

5. *Os axiomas* Φ_H:

   - *Para cada* $s \in S$, $(\forall x_s{:}X_s).apply_{s\to s}(k_s, x_s) = x_s$;
   - *Para cada* $s' \to s \in Ar[\Sigma]$ *e para cada* $f_{i,s'}$ *um axioma:*

     $$(\forall x : X).apply_{s'\to s}(f_{i,s'}(x_1, \ldots, c_i, \ldots, x_n), x_{s'}) = f(x_1, \ldots, apply_{s'\to s_i}(c_i, x_{s'}), \ldots, x_n).$$

Proposição 3.5.17. [BH96, Proposição 7.13] *Considere-se a especificação* HID$[\Sigma, Obs]$ *definida como em cima e o seguinte conjunto de contextos:*

$$\mathcal{C}_H = \{apply_{s'\to s}(ctx, z_{s'}) | s' \to s \in Ar[\Sigma], s \in Obs \text{ e } ctx \text{ variável arbitrária mas fixa do género } s_{s'\to s}\},$$

e seja HBeh$[\mathcal{C}_H]$ *a seguinte família de* Σ_H*-fórmulas, definida para cada* $s \in S$ *da seguinte forma:*

$$\mathrm{HBeh}_s^{\mathcal{C}_\mathcal{H}}(x_s, y_s) = \bigwedge_{c_H \in \mathcal{C}_\mathcal{H}(s)} \forall Var(c_H) c_H[x_s] = c_H[y_s].$$

Tem-se $(\mathrm{HID}[\Sigma, Obs], \mathrm{HBeh}^{\mathcal{C}_H})$ *é uma axiomatização finitária da igualdade observacional* $\approx_{Obs}$ *relativamente a qualquer classe* $\boldsymbol{C}$ *de* Σ*-álgebras.*

Observe-se que nesta axiomatização os contextos são encarados como elementos de novos géneros construídos no método. Tem-se assim que, caso se pretenda proceder à verificação de propriedades observacionais numa especificação por este método, a complexidade da construção destes géneros é a mesma que se encontra, por exemplo, numa prova de propriedades observacionais via indução sobre contextos (ver Secção 4) sendo, portanto, o interesse destas axiomatizações finitárias puramente teórico. Apresentam-se de seguida dois exemplos "extremos"da construção de axiomatizações finitárias por este método:

Exemplo 3.5.18. *Sejam* $\Sigma = (\{s, v\}, \{f_n : s \to v | n \in \mathbb{N}\})$ *uma assinatura,* $Obs = \{v\}$ *um conjunto de géneros observáveis e* $SP = \langle \Sigma, \emptyset \rangle$ *uma especificação. Tem-se*

obviamente que a igualdade observacional $\approx_v$ não é axiomatizável de forma finitária. Seguindo o método descrito na proposição anterior tem-se que,

$$\mathcal{C}_\Sigma = \{f_i(z_s)|i \in \mathbb{N}\} \cup \{z_s, z_v\},\ e$$

$$\mathcal{C}_\Sigma^{Obs} = \{f_i(z_s)|i \in \mathbb{N}\} \cup \{z_v\},$$

pelo que, por 1.,

$$A[\Sigma] = \{s \to v, s \to s, v \to v\},$$

e de acordo com 2.,

$$S_H = \{s, v\} \cup \{s_{[s\to v]}, s_{[s\to s]}, s_{[v\to v]}\}.$$

De acordo com 3. obtém-se:

$$\Omega_H = \{f_n : s \to v|n \in \mathbb{N}\} \cup \{k_s, k_v\} \cup$$

$$\{apply_{v\to v} : s_{v\to v}v \to v, apply_{s\to s} : s_{[s\to s]}s \to s, apply_{s\to v} : s_{[s\to v]}s \to v\}.$$

Para $\mathcal{R}_H = (S_\mathcal{R}, \Omega_\mathcal{R})$, tem-se que

$$S_\mathcal{R} = S_H \setminus S = \{s_{[s\to v]}, s_{[s\to s]}, s_{[v\to v]}\},\ e$$

$$\Omega_\mathcal{R} = \{k_s, k_v\} \cup \{f_{i1,s}|i \in \mathbb{N}\},$$

e para Φ_H tem-se o seguinte conjunto de axiomas:

- $(\forall x_v{:}X_v).apply_{v\to v}(k_v, x_v) = x_v$;
- $(\forall x_s{:}X_s).apply_{s\to s}(k_s, x_s) = x_s$;
- *para cada* $i \in \mathbb{N}$ $(\forall x_s{:}X_s).apply_{s\to v}(f_{i1,s}(z_s), x_s) = f_i(x_s)$.

De acordo com a proposição anterior, tem-se que

$$\mathcal{C}_H = \{apply_{v\to v}(ctx, z_{s'}), ctx \in s_{[v\to v]}\} \cup \{apply_{s\to v}(ctx, z_{s'}), ctx \in s_{[s\to v]}\},$$

que como se pode observar "gera o conjunto"

$$\{z_v\} \cup \{f_i(z_s)|i \in \mathbb{N}\},$$

que não é mais do que $\mathcal{C}_\Sigma^{Obs}$. Então, a seguinte família de equações axiomatiza a igualdade observacional:

$Ax_{\mathrm{HBeh}^{\mathcal{C}_H}}$:

$$[x \sim_s y_s \Leftrightarrow (\forall ctx{:}s_{s\to v}).apply_{s\to v}(ctx, x_s) = apply_{s\to v}(ctx, y_s)],$$

$$[x \sim_v y_v \Leftrightarrow (\forall ctx{:}s_{v\to v}).apply_{v\to v}(ctx, x_v) = apply_{s\to v}(ctx, y_v)].$$

Considere-se agora o caso em que se tem a assinatura $\Sigma = (\{v\}, \{f, g : v \to v\})$ *com o conjunto de géneros observáveis* $Obs = \{v\}$. *Tem-se agora que,*

$$\mathcal{C}_\Sigma = \{h_1(h_2(\cdots h_n(z_v)))|i \in \mathbb{N}, h_i \in \{f, g\}\} \cup \{z_v\} = \mathcal{C}_\Sigma^{Obs},$$

pelo que, por 1.,

$$A[\Sigma] = \{v \to v\},$$

e de acordo com 2.,

$$S_H = \{v\} \cup \{s_{[v\to v]}\}.$$

De acordo com 3. obtém-se:

$$\Omega_H = \{f, g : v \to v\} \cup \{k_v\} \cup \{apply_{v\to v} : s_{[v\to v]}v \to v\}.$$

Para $\mathcal{R}_H = (S_\mathcal{R}, \Omega_\mathcal{R})$, *tem-se que*

$$S_\mathcal{R} = s_{[v\to v]},$$

$$\Omega_\mathcal{R} = \{k_v\} \cup \{f_{1,s}, g_{1,s}\},$$

e para $\mathcal{A}x_H$ *tem-se o seguinte conjunto de axiomas:*

- $(\forall x_v{:}X_v).apply_{v\to v}(k_v, x_v) = x_v$;
- $(\forall x_s{:}X_s).apply_{v\to v}(f_{1,s}(z_s), x_s) = f(x_s)$;
- $(\forall x_s{:}X_s).apply_{v\to v}(g_{1,s}(z_s), x_s) = g(x_s)$.

De acordo com a proposição anterior, tem-se que

$$\mathcal{C}_H = \{apply_{v\to v}(ctx, z_{s'}), ctx \in s_{[v\to v]}\},$$

que como se pode observar "gera o conjunto"[7] $\mathcal{C}_\Sigma^{Obs}$. *Então, a seguinte equação axiomatiza a igualdade observacional:*

$Ax_{\mathrm{HBeh}^{\mathcal{C}_H}}$:

$$[x_v \sim_v y_v \Leftrightarrow (\forall ctx{:}s_{v\to v}).apply_{v\to v}(ctx, x_v) = apply_{s\to v}(ctx, y_v)].$$

◇

[7] *Note-se que termos do tipo* $f(g(\cdots(x_v)))$ *também são entendidos como variáveis do sort* $s_{v\to v}$, *uma vez que são resultado de aplicações das funções* $\{f_{1,s}, g_{1,s}\}$, *que por definição são de aridade* $s_{v\to v}v \to s_{v\to v}$.

Em [BH96], o estudo da axiomatização finitária da igualdade observacional também é feito para o caso parcial. Por forma a contornar a infinitaridade provocada pela verificação da definibilidade dos termos (ver Exemplo 3.5.8), introduzem-se novos símbolos de predicado $(D_H)_s$ axiomatizados (se necessário por novos símbolos de função). Para o caso da axiomatização geral da igualdade observacional parcial, é proposta a codificação dos termos de $T_\Sigma(X_{In})$ na parte escondida, especificando a sua aplicação por novos símbolos de função da parte escondida.

3.6 Sistema de Prova para Especificações Estruturadas com Operadores Comportamentais

Estende-se nesta secção o sistema de prova para especificações estruturadas apresentado na Secção 2.5 ao caso das especificações estruturadas com operadores comportamentais. Este sistema é apresentado em [Hen97], havendo outros sistemas similares na literatura adequados à escolha de outros operadores.

Definição 3.6.1. *O sistema de prova para especificações estruturadas apresentado na Definição 2.5.3 é estendido ao trabalho com especificações com* reachability constraints *pela introdução dos seguintes axiomas e regras de inferência:*

$$\textbf{\textit{reach}}\ SP\ \textbf{\textit{with}}\ \Omega_{\mathcal{R}} \vdash GEN_s \text{ para todos os } s \in Sort(Sig(SP))$$

$$\frac{SP \vdash \phi}{\textbf{\textit{reach}}\ SP\ \textbf{\textit{with}}\ \Omega_{\mathcal{R}} \vdash \phi}$$

$$\frac{SP \vdash \phi_i, \{\phi_i | i \in I\} \vdash \phi}{SP \vdash \phi}$$

Definição 3.6.2. *Sejam $\langle \mathcal{L}(\Sigma), AX[\sim]\rangle$ especificação da igualdade observacional $\approx_{Obs,In}$ (Definição 3.5.7), e* HID $+ \langle \mathcal{L}(\Sigma), FinAX[\sim]\rangle$ *a especificação finitária com parte escondida de $\approx_{Obs,In}$ relativamente a $[\![SP]\!]$. O sistema de prova para especificações estruturadas apresentado na Definição 2.5.3 é estendido ao trabalho com especificações com operadores observacionais pela introdução dos seguintes axiomas:*

$$\frac{SP \vdash \phi}{(\textbf{\textit{behaviour}}\ SP\ \textbf{\textit{wrt}}\ \approx_{Obs,In}) + \langle \mathcal{L}(\Sigma), AX[\sim]\rangle \vdash \mathcal{L}(\phi)}$$

$$\frac{(\textbf{\textit{behaviour}}\ SP\ \textbf{\textit{wrt}}\ \approx_{Obs,In}) + \langle \mathcal{L}(\Sigma), AX[\sim]\rangle \vdash \phi}{(\textbf{\textit{behaviour}}\ SP\ \textbf{\textit{wrt}}\ \approx_{Obs,In}) \vdash \phi}$$

$$\frac{SP + \langle \mathcal{L}(\Sigma), AX[\sim] \rangle \vdash \mathcal{L}(\phi)}{SP/\approx_{Obs,In} \vdash \phi}$$

$$\frac{SP + \mathrm{HID} + \langle \mathcal{L}(\Sigma), FinAX[\sim] \rangle \vdash \mathcal{L}(\phi)}{SP/\approx_{Obs,In} \vdash \phi}$$

O Teorema que se segue estabelece a correcção e adequação do sistema de prova em cima definido:

Teorema 3.6.3. [Hen97, Apêndice F] *Seja SP uma especificação sobre a assinatura Σ e ϕ uma Σ-álgebra. Tem-se que*

$$SP \models \phi \text{ sse } SP \vdash \phi.$$

Capítulo 4

O Caso *flat* Observacional - Outras Abordagens

Tal como já foi referido, o estudo da semântica observacional de sistemas de software tem um grande potencial na aplicação das práticas de análise e desenvolvimento de programas. Por esta razão, existem na literatura vários trabalhos com o objectivo de criar algoritmos eficientes para a verificação observacional de propriedades e outros no sentido de abordar o paradigma a um nível mais abstracto, por forma a analisá-lo e estudá-lo com recurso a conhecimentos de áreas mais teóricas da matemática. Apresentam-se nesta Secção duas abordagens a este assunto: uma primeira, de orientação mais prática desenvolvida por *J. Goguen* e seus seguidores (Secção 4.1) e, uma segunda mais teórica, desenvolvida por *M. Martins* e *D. Pigozzi*, na qual se aborda o assunto com base nos resultados da *Lógica Algébrica Abstracta* (Secção 4.2).

Efectivamente, o contributo das abordagens escolhidas é muito significativo, na medida em que, por exemplo, em resultado dos trabalhos do grupo de *Goguen*, surgiram alguns dos mais eficientes algoritmos de verificação observacional disponíveis até ao momento e, dos da *Lógica Algébrica Abstracta*, uma compreensão do assunto a um nível mais abstracto, obtido pelo recurso a conceitos de cariz mais teórico (por exemplo, a utilização das propriedades da *congruência de Leibniz*). Contudo, existem outros trabalhos e abordagens que poderiam figurar nesta escolha, como são os exemplos das abordagens de *R. Diaconescu* e *K. Futatsugi* [DF00] e de *A. Bohoula* e *M. Rusinowitch* (cf. [BBR98]). A abordagem de *R. Diaconescu* e *K. Futatsugi* é muito próxima da apresentada na Secção 4.1. Os algoritmos desenvolvidos por este grupo

estão implementados nas ferramentas de verificação da linguagem de especificação CafeOBJ (ver http://www.ldl.jaist.ac.jp/cafeobj/). A abordagem de *A. Bohoula* e *M. Rusinowitch*, distingue-se das abordagens de *Goguen* e *Diaconescu*, na medida em que se apoia no princípio da *indução contextual* ou *indução estrutural sobre os contextos*, contrariamente às anteriores, que se apoiam nos princípios da coindução (Secção 4.1.2). A *indução contextual* é um princípio de verificação observacional de propriedades introduzido em [Hen90] que tem por base o conceito de *indução estrutural*:

Princípio da indução contextual:

Sejam $\Sigma = (S, \Omega)$ uma assinatura heterogénea, $Obs \subseteq S$ um conjunto de géneros observáveis e $t_i \in T_\Sigma(X)_{s_i}$ para $i \in \{1, \ldots, n\}$. A propriedade $P(c)$ verifica-se para todos os contextos $c \in \mathcal{C}_\Sigma^{Obs}$ se:

1. $P(z_s)$ se verifica em todos os $s \in Obs$;

2. Para todos os contextos da forma $f(t_1, \ldots, t_{i-1}, c, t_{i+1}, \ldots, t_n)$ tais que $f : s_1, \ldots s_n \rightarrow s \in \Sigma, s \in Obs$ e c contexto sobre Σ do género s_i, tem-se que

$$P(f(t_1, \ldots, t_{i-1}, c, t_{i+1}, \ldots, t_n))$$

   sempre que se verifica $P(c')$ para todos os subcontextos observáveis c' de c.

Desta forma, para se provar a igualdade observacional de dois elementos a, b do género s, prova-se que a propriedade $c[a] = c[b]$ é verdade para todos os contextos de $\mathcal{C}_\Sigma^{Obs}(s)$. Por forma a efectuar este tipo de prova eficientemente, *A. Bohoula* e *M. Rusinowitch* recorrem a algumas técnicas de verificação indutiva por eles anteriormente investigadas, por exemplo, em [BR95, Bou97]. Destes trabalhos surgem os algoritmos de verificação observacional publicados em [BBR98, BR02], e posteriormente implementados na ferramenta de verificação observacional SPIKE (ver http://www.loria.fr/equipes/cassis/softwares/spike/).

4.1 Abordagem de Goguen

Segundo a Definição 3.1.1, todos os contextos construídos a partir de todos os símbolos de operação de Σ e pelas variáveis de Input são considerados na definição de Σ-igualdade observacional parcial. Noutras abordagens à observabilidade, como são os casos das abordagens dos grupos de *J. Goguen* e de *R. Diaconescu* (cf. [GM00, GM99, Roş00, DF02]), apenas um subconjunto Γ dos símbolos de operação de Σ, denominado por

conjunto das operações congruentes, é utilizado na definição dos contextos a considerar em $\approx_{Obs}^{\Gamma}$. Grande parte dos métodos de prova disponíveis para a verificação de propriedades observacionais (nomeadamente os apresentados na Secção 4.1.2), foram desenvolvidos segundo este formalismo, razão pela qual se apresentam na Secção 4.1.1 algumas definições e notação relativas a esta abordagem.

4.1.1 Alguma notação e terminologia

Tal como foi referido, segundo algumas abordagens, apenas alguns símbolos de função são considerados para a construção do conjunto de contextos observáveis. Este princípio está na base do conceito de Γ-*Igualdade observacional*:

Definição 4.1.1 (Γ-Igualdade observacional)**.** *Sejam $\Sigma = (S, \Omega)$ uma assinatura, $Obs \subseteq S$ conjunto de géneros observáveis, A uma Σ-álgebra e Γ uma subassinatura de Σ. Define-se* Γ-Igualdade Observacional em A segundo *Obs como sendo a relação $\approx_{\mathcal{C}_{\Gamma}^{Obs}}^{A}$, isto é, como sendo a igualdade contextual total em A induzida pelo conjunto dos* Γ-contextos observáveis. *A relação de Γ-igualdade observacional em A denota-se por $\approx_{Obs}^{\Gamma\ A}$. Define-se também $\approx_{\Gamma}^{Obs}$ como sendo a família de relações $(\approx_{\Gamma}^{Obs})_{A \in Alg(\Sigma)}$.*

De forma análoga, definem-se numa álgebra A os conceitos de Γ-*congruência* e Γ-*congruência escondida*. Segundo esta abordagem, uma especificação (*flat*) $SP = \langle \Sigma, \Phi \rangle$, é também munida por uma subassinatura Γ de Σ, podendo ser representada na forma $SP = \langle (\Sigma, \Gamma), \Phi \rangle$ (a especificação $SP = \langle \Sigma, \Phi \rangle$ com subassinatura de operações congruentes Γ).

Este formalismo é, segundo os autores, a forma mais natural de lidar com o *não determinismo de especificações*. Este não determinismo é apontado como um aspecto inerente à especificação algébrica de programas e refere-se ao grau de liberdade da interpretação das operações numa especificação. Observe-se que o não determinismo de uma especificação SP não tem a ver com o determinismo das operações dos seus modelos, mas sim com o determinismo da especificação na escolha dos modelos, isto é, tem-se que dado um modelo $P \in [\![SP]\!]$, a escolha de uma função f^P tal que $f \in \Sigma \setminus \Gamma$, faz-se em absoluta liberdade aos requisitos da especificação (desde que obedeça à natureza dos géneros), sendo que f^P tem de ser uma função determinista. É também natural a presença deste conceito no processo de refinamento (ver Secções 2.4 e 3.4), onde em cada passo do processo se incrementa o grau de determinismo da especificação inicial. Este assunto é apresentado, por exemplo, em [GM99].

A introdução da subassinatura Γ de operações congruentes, requer uma adaptação no conceito de sistema de reescrita (Secção 2.6), por forma a ser possível reescrever correctamente termos com operações não congruentes (em $\Gamma \setminus \Sigma$). Desta adaptação resultam os sistemas de *reescrita* Γ-*observacionais* apresentados, por exemplo, em [GLR00a]. Define-se de seguida a relação de reescrita Γ-observacional:

Definição 4.1.2 (Relação de reescrita Γ-observacional)**.** *Seja* $SP = \langle(\Sigma, \Gamma), \Phi\rangle$ *uma especificação e Obs o conjunto de géneros observáveis da assinatura* Σ*. A* relação de reescrita Γ-observacional associada ao sistema de reescrita $R(SP)$ com respeito a *Obs é a mais pequena relação* $\rhd_\Gamma$ *tal que:*

- $\alpha(l) \rhd_\Gamma \alpha(r)$ *para toda a regra* $l \rhd r \in R(SP)$ *e para toda a valoração* $\alpha : X \to T_\Sigma(X)$*;*
- *Se* $t \rhd_\Gamma t'$ *e* $t, t' \in T_\Sigma(X)_s$ *para* $s \in Obs$*, então* $\sigma(W, t) \rhd_\Gamma \sigma(W, t)$ *para todo o* $\sigma \in Der(\Sigma)$*;*
- *Se* $t \rhd_\Gamma t'$ *e* $t, t' \in T_\Sigma(X)_s$ *para* $s \in S \setminus Obs$*, então* $\delta(W, t) \rhd_\Gamma \delta(W, t)$ *para todo o* $\delta \in \Gamma$*.*

Em [GLR00a, Secção 3.2], é apresentado um algoritmo que adapta os verificadores automáticos que suportem reescrita standard à reescrita Γ-observacional. Este algoritmo é denominado por $Paint_{\Sigma,\Gamma}$ e está actualmente implementado no verificador BOBJ.

Todos os métodos que se trabalham nesta Secção são para a verificação $\models_{\approx_{Obs}^{\Gamma}}$, sendo a sua particularização ao caso $\models_{\approx_{Obs}}$ directa (caso se faça $\Gamma = \Sigma$).

Uma outra diferença entre a abordagem destes autores e a abordagem central do texto tem a ver com o conceito de modelo observacional de uma especificação. Enquanto que no presente texto se entende por $SP \models_{\approx_{obs}} \phi$ a satisfação $[\![SP]\!] \models_{\approx_{Obs}} \phi$, estes autores entendem $[\![\textbf{behaviour } SP \textbf{ wrt } \approx_{Obs}]\!] \models_{\approx_{Obs}} \phi$.

4.1.2 Métodos de Verificação Γ-Observacional

Apresentam-se nesta Secção alguns algoritmos de prova de propriedades observacionais implementados em algumas ferramentas de verificação, nomeadamente nos verificadores da família OBJ.

Hidden Coinduction

O Teorema 3.1.21 está na base de grande parte dos métodos de verificação observacional de sistemas, nomeadamente nos métodos coindutivos implementados nos verificadores de teoremas observacionais BOBJ e CafeOBJ [NSF]. O Teorema que se segue generaliza o Teorema 3.1.21 ao caso das Γ-congruências:

Teorema 4.1.3. *Sejam Σ uma assinatura, Γ uma subassinatura de Σ e A uma Σ-álgebra. Tem-se que a Γ-igualdade observacional em A é a maior Γ-congruência escondida nessa álgebra.*

Assim, para provar a Γ-igualdade observacional entre dois elementos de uma Σ-álgebra, basta encontrar uma Γ-congruência escondida nessa álgebra e provar que esses elementos são congruentes por essa relação:

Algoritmo 1 (Hidden Coinduction)**.** *Sejam $\Sigma = (S, \Omega)$ uma assinatura, Γ uma subassinatura de Σ, $Obs \subseteq S$ um conjunto de géneros observáveis e A uma Σ-álgebra. Para provar que $a {\approx}^{\Gamma}_{Obs}{}^{A} a'$, para $a, a' \in A_s, s \in S$, fazer:*

1. *Definir uma relação binária $\approx_{cand}$ em A;*
2. *Mostrar que $\approx_{cand}$ é uma Γ-congruência escondida em A;*
3. *Mostrar que $a \approx_{cand} a'$;*

A relação $\approx_{cand}$ é denominada por relação candidata.

Exemplo 4.1.4 (FLAGS [GM99])**.** *Considere-se a especificação* `FLAGS` *apresentada no Exemplo 3.1.10. Dada uma Σ_{FLAGS}-álgebra $A \in [\![\texttt{FLAGS}]\!]$, pretende-se provar por* Hidden Coinduction *que $A \models_{\approx^{\Gamma}_{Obs}} \texttt{rev(rev(x))=x}$, para $\Gamma = \Sigma_{FALGS}$. Defina-se a relação binária θ da seguinte forma: para todos os elementos $\texttt{f}_1, \texttt{f}_2 \in A_{\texttt{flag}}$, para todos $\texttt{b}_1, \texttt{b}_2 \in A_{\texttt{bool}}$:*

$$\texttt{f}_1 \theta_{flag} \texttt{f}_2 \text{ sse } \texttt{?up}^A\texttt{(f}_1\texttt{)} = \texttt{?up}^A\texttt{(f}_2\texttt{)}$$

$$\texttt{b}_1 \theta_{bool} \texttt{b}_2 \text{ sse } \texttt{b}_1 = \texttt{b}_2.$$

Tem-se que θ é uma relação de congruência escondida em A, uma vez que coincide com a relação $=$ no género observável `bool` *e é congruente com as operações de* `FLAGS` *(Exemplo 3.5.13). Portanto $A \models_{\approx^{\Gamma}_{Obs}} \texttt{rev(rev(x))=x}$.*

◇

No Exemplo 4.1.4 foi possível definir uma relação candidata em A exclusivamente à custa dos atributos da especificação, isto é, à custa de funções de resultado observável com um único elemento não observável. Muitas vezes isto não é possível e, por forma a conseguir definir a relação candidata, estende-se a especificação de forma adequada. Naturalmente, esta extensão tem que ser conservativa, no sentido em que $Sig(SP) \subseteq Sig(SP')$ e que para todo o modelo $A \in [\![SP]\!]$, exista $A' \in [\![SP']\!]$ tal que $A' \restriction_{Sig(SP)} = A$. O exemplo que se segue ilustra uma extensão deste tipo:

Exemplo 4.1.5 (STREAM ([Roş00])). *Considere-se a especificação* `STREAM` *do Exemplo 3.1.11. Pretende-se provar* $\texttt{STREAM} \models^{\Gamma}_{\approx_{Obs}} (\forall \texttt{s:stream}).\texttt{head(s)::tail(s)=s}$. *Tem-se intuitivamente que duas* streams *são observacionalmente iguais se tiverem os mesmos elementos pela mesma ordem. Contrariamente ao que acontece no exemplo anterior, não é possível definir uma relação candidata exclusivamente à custa de atributos. Por exemplo, a relação* $\texttt{s} \approx_{cand} \texttt{s'}$ *sse* `head(s)=head(s')`*, não é congruência escondida, uma vez que* `head(s)=head(s')` *não implica que* `head(tail(s))=head(tail(s'))`. *Por forma a conseguir denotar os contextos* $\texttt{head(tail(}\cdots\texttt{(tail(s))))}$*, considere-se a seguinte extensão da especificação* `STREAM`*:*

```
Spec STREAM_headn=enrich STREAM + NAT by

[OP]
    headn: nat,stream -> elt;
[AX]

(∀l,l':stream).headn(0,l)=head(l);

(∀l,l':stream,∀n:nat).headn(s(n),l)=headn(n,tail(l));
```

Donde se deduz directamente que:

1. `headn(0,e::l)=e;`

2. `headn(s(n),e::l)=headn(n,l);`

Considere-se agora uma Σ_{STREAM}*-álgebra* $A \in [\![\texttt{STREAM}]\!]$ *arbitrária e uma relação* θ^A *definida como sendo a igualdade estrita para os géneros* `elt` *e* `nat`*, e definida para* `stream` *como:*

$$\texttt{l}\,\theta^A_{stream}\,\texttt{l'} \text{ sse para todo o } \texttt{n} \in \texttt{nat}, \texttt{headn}^A\texttt{(n,l)} = \texttt{headn}^A\texttt{(n,l')}.$$

Seja por hipótese $\texttt{l}\theta^A_{stream}\texttt{l'}$. *Pela definição de* θ^A_{stream}, *tem-se que para todo o* $\texttt{n} \in \texttt{nat}$, $\texttt{headn}^A\texttt{(n,l)} = \texttt{headn}^A\texttt{(n,l')}$, *e assim* $\texttt{headn}^A\texttt{(s}^A\texttt{(n),l)} = \texttt{headn}^A\texttt{(s}^A\texttt{(n),l')}$, *donde para todo o* $\texttt{n} \in \texttt{nat}$

$$\texttt{headn}^A\texttt{(n,tail}^A\texttt{(l))} = \texttt{headn}^A\texttt{(n,tail}^A\texttt{(l'))},$$

ou seja, $\texttt{tail}^A\texttt{(l)}\theta^A_{stream}\texttt{tail}^A\texttt{(l')}$. *Tem-se assim que* θ^A_{stream} *é congruência escondida. Uma vez que* $\texttt{STREAM}_{headn}$ *é extensão conservativa de* STREAM, *só falta provar que*

$$\texttt{head}^A\texttt{(l)::}^A\texttt{tail}^A\texttt{(l)}\theta^A_{stream}\texttt{l}.$$

Caso n=0, *tem-se por (1) que*

$$\texttt{headn}^A\texttt{(0,head}^A\texttt{(l)::}^A\texttt{tail}^A\texttt{(l))} = \texttt{head}^A\texttt{(l)=headn(0,l)}.$$

Por outro lado, tem-se também que

$$\texttt{headn}^A\texttt{(s}^A\texttt{(n),head}^A\texttt{(l)::}^A\texttt{tail}^A\texttt{(l))} = \texttt{headn}^A\texttt{(n,tail}^A\texttt{(l))} = \texttt{headn}^A\texttt{(s}^A\texttt{(n),l)},$$

e assim, tem-se por indução que $\texttt{head}^A\texttt{(l)::}^A\texttt{tail}^A\texttt{(l)}\theta^A_{stream}\texttt{l}$. *Uma vez que A arbitrária, chega-se a* $\texttt{STREAM} \models_{\approx^{\Gamma}_{Obs}} \texttt{(}\forall\texttt{l:stream)}\texttt{.head(l)::tail(l)=l}$.

$\diamond$

O método coindutivo aqui descrito aparece na literatura como *hidden coinduction* e pode ser encontrado, por exemplo, em [GM00, GM99]. Contudo, este método ainda requer intervenção humana na escolha da relação candidata $\approx_{cand}$. A automatização do processo de coindução tem vindo a ser estudada ao longo dos tempos, tendo os trabalhos do grupo de *J. Goguen* um papel de grande relevância no que respeita à construção de métodos eficientes para este tipo de prova. Dos seus trabalhos, resultaram os métodos da *Hidden Coinduction*, *Behavioural Coinductive Rewriting*, *Circular Coinductive Rewriting* e por fim o *Conditional Circular Coinductive Rewriting with Case Analysis* segundo os próprios autores: *"(...)seems to be the most powerful automated proof technique now available for behavioral equivalence(...)"*(in [GLR02]).Note-se que esta eficiência não se refere apenas à rapidez ou complexidade algorítmica dos métodos, mas também à gama de casos verificáveis por cada um, na medida em que, tal como se apresenta no Capítulo 5, não existem algoritmos que verifiquem todos os teoremas observacionais em todas as especificações. Em [Roş00, Capítulo IV], pode ser encontrada uma apresentação retrospectiva do aparecimento desta família de algoritmos (até ao Δ-coindução circular *rewriting*), sendo este factor bem explorado.

A Δ-Coindução

Tal como foi referido anteriormente, o processo coindutivo apresentado em cima, necessita de intervenção humana na escolha da relação candidata. Os primeiros passos dados na automatização deste método foram no sentido de procurar um conjunto (finito) de contextos Δ que de alguma maneira gerasse todo o conjunto $\mathcal{C}_\Gamma^{Obs}$, por forma a induzir uma relação de congruência escondida $\approx_{Obs}^{\Delta}$ tal que, dada um Σ-álgebra A, ${\approx_{Obs}^{\Delta}}^A \leq {\approx_{Obs}^{\Gamma}}^A$. A definição que se apresenta de seguida, aparece inicialmente em [BH99] como *conjunto completo de contextos*:

Definição 4.1.6 (Conjunto completo de observadores)**.** *Sejam Σ assinatura e $\Delta \subseteq \mathcal{C}_\Sigma$ um conjunto de contextos de Σ. Δ diz-se* conjunto completo de observadores*, se e só se para todo o contexto observável $c \in \mathcal{C}_\Sigma^{Obs}$, existe um subcontexto de c em Δ.*

São exemplos de conjuntos completos de observadores da assinatura Σ_{STREAM} (Exemplo 3.1.11), os seguintes conjuntos:

$$\Delta^0 = \{\texttt{head(z}_{stream}\texttt{),tail(z}_{stream}\texttt{)}\}$$

$$\Delta^1 = \{\texttt{head(tail(z}_{stream}\texttt{)),tail(tail(z}_{stream}\texttt{))}\}$$

$$\vdots$$

$$\Delta^n = \mathcal{C}_\Sigma^{Obs}$$

Para o mesmo propósito, define-se em [RG00, Definição 14], o conceito de *cobase forte de coindução*[1], generalizado posteriormente em [GR99b, Definição 2.6] pelo *conceito de cobase de coindução*:

Definição 4.1.7 (Cobase de coindução)**.** *Sejam $SP = \langle(\Sigma, \Gamma), \Phi\rangle$ uma especificação, $Obs \subseteq S$ um conjunto de géneros observáveis (para S conjunto de géneros de Σ), $SP' = \langle(\Sigma', \Gamma'), \Phi'\rangle$ uma extensão conservativa de SP, e $\Delta \subseteq \Sigma'$. O conjunto Δ* é uma cobase para *SP, se para todos os termos $t, t' \in T_\Sigma(X)_s$ tais que $s \in S \setminus Obs$ se tem que, se para todo $\delta \in \Delta$ apropriado, se verifica $SP \models_{\approx_{Obs}^{\Gamma}} (\forall X)(\forall Var(\gamma)).\delta(t)\texttt{=}\delta(t')$, então $SP' \models_{\approx_{Obs}^{\Gamma'}} (\forall X).t\texttt{=}t'$.*

A Proposição que se segue, estabelece uma forma de construir cobases Δ para o processo de Δ-coindução:

[1] Aparece inicialmente denominado por *Cobase de coindução*, na medida em que o conceito mais geral só é definido posteriormente;

Proposição 4.1.8. *Todo o conjunto completo de observadores é uma cobase.*

A demonstração da proposição baseia-se em dois importantes resultados: *toda a cobase forte é cobase* e *todo o conjunto completo de contextos é uma cobase forte* (cf. [Roş00, Teorema 41 e Proposição 39]). O resultado anterior pode parecer pouco natural, na medida em que na Definição 4.1.6, Δ aparece como um sendo um conjunto de Γ-contextos de Σ, e na Definição 4.1.7 como uma subassinatura de Σ'. Este facto compreende-se facilmente se se considerar para a especificação $SP = \langle(\Sigma, \Gamma), \Phi\rangle$ a especificação $SP' = \langle(Der(\Sigma), \Gamma'), \Phi'\rangle$ (Definição 2.1.9).

Observe-se que a definição anterior induz um método de prova observacional, na medida em que, naquelas condições, para provar que $t \approx_{Obs}^{\Gamma'} t'$ se verifica, basta provar que $\delta(t) \approx_{Obs}^{\Gamma} \delta(t')$ se verifica para quaisquer variáveis de δ e para todos os $\delta \in \Delta$ apropriados. Este princípio de prova denomina-se por *Princípio da Δ-coindução*. Note-se que a Δ-coindução pode ser vista como uma generalização do método da coindução apresentado no Algoritmo 1, na medida em que este método é o caso particular onde $SP = SP'$, $\Delta = \Gamma$ e $\approx = \{(a, a') | \delta(a_1, \ldots, a_n, a) = \delta(a_1, \ldots, a_n, a'),$ para todos os $\delta \in \Gamma, a_1, \ldots, a_n \in A$ (apropriados)$\}$. Uma vez que as operações em Γ são por definição congruentes, tem-se que $\approx$ é uma congruência escondida e, pelo Teorema 4.1.3, que $\approx_\Delta \leq \approx_\Gamma$, tal como se pretendia. Várias caracterizações do conceito de cobase tem vindo a ser estudadas, nomeadamente as baseadas no conceito da equivalência semântica de especificações (para $SP' = \langle(Der(\Sigma), \Gamma'), \Phi'\rangle$) (cf. [Roş00, Teorema 46]) existindo alguns métodos automáticos para a construção de cobases de coindução (ver, por exemplo, o algoritmo *Test Set Coinduction* implementado no CafeObj, [Mat98, Capítulo 4]).

O Algoritmo que se segue foi inicialmente apresentado em [Ros99, Secção 6.1] e implementa o princípio da Δ-coindução recorrendo à *reescrita observacional* (Secção 4.1.1):

Algoritmo 2 (*Behavioural Coinductive Rewriting*)**.** *O algoritmo recebe como INPUT uma especificação $SP = \langle(\Sigma, \Gamma), \Phi\rangle$, a indicação de um conjunto de géneros observáveis de Σ, uma cobase Δ e um par de $Sig(SP)$-termos (t, t'). Devolve a resposta **SIM** caso $t \approx_{Obs}^{\Gamma} t'$, ou um conjunto de pares $\mathcal{G}$ no caso contrário.*

1. *Fazer $\mathcal{G} = (t, t')$;*

2. *Reescrever Γ-observacionalmente todos os pares de $\mathcal{G}$ (pelo algoritmo $Paint_{\Sigma,\Gamma}$);*

3. *Remover de* $\mathcal{G}$ *todos os pares* (r, r') *tais que* $r = r'$*;*

4. *Se* $\mathcal{G} = \emptyset$ *então devolver* ***SIM****;*

5. *Se* $\mathcal{G}$ *não possui termos não observáveis com raiz em* Δ *então devolver* $\mathcal{G}$

6. *Expandir os pares de* $\mathcal{G}$ *por* Δ*;*

7. *ir para 2.*

Exemplo 4.1.9 (STREAM ([RG01]))**.** *Provou-se por coindução no Exemplo 4.1.5 que* $\texttt{STREAM} \models_{\approx_{Obs}^{\Gamma}} (\forall \texttt{s:stream}).\texttt{head(s)::tail(s)=s}$*. Faz-se agora a mesma prova por* Δ*-coindução. Tem-se que* $\Delta = \{\texttt{head}(\texttt{z}_{stream}), \texttt{tail}(\texttt{z}_{stream})\}$ *é conjunto completo de observadores, e portanto, pelo Lema 4.1.8, também é uma cobase. Assim, uma vez que*

$$\texttt{head(head(s)::tail(s))=head(s)}$$

e

$$\texttt{tail(head(s)::tail(s))=tail(s)},$$

tem-se que

$$\texttt{STREAM} \models_{\approx_{Obs}^{\Gamma}} (\forall \texttt{s:stream}).\texttt{head(s)::tail(s)=s},$$

tal como se pretendia.

◇

Observe-se agora o exemplo que se segue:

Exemplo 4.1.10 (STREAM ([RG01]))**.** *Considere-se a especificação* $\texttt{STREAM}_{merge}$*, obtida pelo enriquecimento de* STREAM *(Exemplo 3.1.11) pelas funções* odd, even *e* merge*, onde as duas primeiras, dada uma* stream s*, devolvem uma nova* stream s' *com os elementos das posições ímpares e pares respectivamente, e a função* merge*, dadas duas* streams

$$\texttt{s=s}_1\texttt{::s}_2\texttt{::s}_3\texttt{::}\cdots \text{ e } \texttt{s'=s'}_1\texttt{::s'}_2\texttt{::s'}_3\texttt{::}\cdots,$$

devolve a stream $\texttt{s}_1\texttt{::s'}_1\texttt{::s}_2\texttt{::s'}_2\texttt{::}\cdots$.

$\texttt{STREAM}_{merge}$=enrich STREAM by

[OP]

```
    odd:stream -> stream;
    even:stream ->  stream;
    merge:stream,stream ->  stream;
```

[AX]

∀s:stream.head(odd(s))=head(s);

∀s:stream.tail(odd(s))=even(tail(s));

∀s:stream.head(even(s))=even(tail(tail(s)));

∀s:stream.tail(even(s))=even(tail(tail(s)));

∀s,s':stream.head(merge(s,s'))=head(s);

∀s:stream.tail(merge(s,s'))=merge(s',tail(s));

Pretende-se mostrar que $\mathtt{STREAM}_{merge} \models_{\approx_{Obs}} \forall \mathtt{s:stream.odd(merge(s,s'))=s}$. *Usando o Algoritmo 2 e a cobase* $\Delta = \{\mathtt{head(z}_{stream}\mathtt{)},\mathtt{tail(z}_{stream}\mathtt{)}\}$, *tem-se que*

$$\mathtt{head(odd(merge(s,s')))=head(merge(s,s'))=head(s)}.$$

Por outro lado, tem-se que

$$\mathtt{tail(odd(merge(s,s')))=even(tail(merge(s,s')))}$$

$$\mathtt{=even(merge(s',tail(s)))}.$$

Uma vez que os termos $\mathtt{even(merge(s',tail(s)))}$ *e* $\mathtt{tail(s)}$ *não coincidem nem são observáveis, repete-se o método:*

$$\mathtt{head(even(merge(s',tail(s))))=head(tail(merge(tail(s),tail(s'))))}$$

$$\mathtt{=head(merge(tail(s,tail(s'))))=head(tail(s))}.$$

Por outro lado

$$\mathtt{tail(even(merge(s',tail(s))))=even(tail(tail(merge(s',tail(s))))}$$

$$\mathtt{=even(tail(merge(tail(s),tail(s'))))}$$

$$\mathtt{=even(merge(tail(s'),tail(tail(s))))}.$$

Observe-se que o método de Δ*-coindução entra aqui em ciclo, uma vez que a continuação deste processo devolve sucessivamente os pares de termos* (u, u') *tais que*

$$\mathtt{u=even(merge(tail(}\cdots\mathtt{tail(s'),tail(}\cdots\mathtt{(tail(tail(s)))))}\ \textit{e}$$

$$\mathtt{u'=tail(}\cdots\mathtt{(tail(tail(s))))}.$$

Obtém-se um ciclo semelhante a este na verificação da propriedade

$$\mathtt{STREAM}_{merge} \models_{\approx_{Obs}} \forall \mathtt{s:stream.merge(odd(s),even(s))=s},$$

para $\Delta = \{\mathtt{head(z}_{stream}\mathtt{)},\mathtt{tail(z}_{stream}\mathtt{)}\}$, *tal como se pode ver em* [RG01].

◇

São estas circularidades que motivaram o estudo do método da $\Delta^{\circlearrowleft}$-coindução.

Coindução Circular ($\Delta^{\circlearrowleft}$-Coindução)

Por forma a resolver os casos de circularidade observados nas provas por Δ-coindução (ver Exemplo 4.1.9), define-se em [RG01] a relação de igualdade observacional circular coindutiva. Este conceito baseia-se na seguinte relação:

Definição 4.1.11 (Igualdade observacional forte)**.** *As funções $\alpha, \beta : X \cup \{Z\} \to T_\Gamma(X)$ dizem-se* observacionalmente equivalentes *sse $\alpha(x) \approx^\Gamma_{Obs} \beta(x)$ para todo o $x \in X$. Dois termos t, t' dizem-se* iguais observacionalmente em sentido forte *sse para toda a Σ-álgebra A, e para quaisquer $\tau, \tau' : X \to A$ tais que $\tau(x) \approx^\Gamma_{Obs} \tau'(x)$ para todo $x \in X$, se tem que $\tau(t) \approx^\Gamma_{Obs} \tau'(t')$. Quando α e β são observacionalmente equivalentes escreve-se $\alpha \asymp \beta$, e quando t e t' são* iguais observacionalmente em sentido forte *escreve-se $t \cong t'$.*

Definição 4.1.12 (Equivalência $\Delta^{\circlearrowleft}$-coindutiva)**.** *Os termos t, t' são $\Delta^{\circlearrowleft}$-coindutivamente equivalentes sse para todo $\delta \in \Delta$ se tem que ou $\delta(W, t) \approx^\Gamma_{Obs} \delta(W, t') \approx^\Gamma_{Obs} u$ para algum Γ-termo u, ou $\delta(W, t) \approx^\Gamma_{Obs} \alpha(c[t])$ e $\delta(W, t') \approx^\Gamma_{Obs} \beta(c[t'])$ para $\alpha \asymp \beta$ e $c < \delta$. Quando dois termos t e t' são $\Delta^{\circlearrowleft}$-coindutivamente equivalentes, escreve-se $t \approx_{\Delta^{\circlearrowleft}} t'$.*

O Teorema que se segue fundamenta o método da Δ-coindução circular:

Teorema 4.1.13.

$$\approx_{\Delta^{\circlearrowleft}} \leq \approx^\Gamma_{Obs}$$

A demonstração do resultado pode ser consultada em [RG01, Teorema 3]. Resolve-se de seguida o problema da circularidade resultante da aplicação do Algoritmo 4.1.2 ao Exemplo 4.1.10:

Exemplo 4.1.14 (STREAM ([RG01]))**.** *No Exemplo 4.1.10, o método da Δ-coindução entrou em ciclo na verificação de* $\texttt{STREAM}_{merge} \models_{\approx_{Obs}} \forall \texttt{s:stream.odd(merge(s,s')=s}$. *Pretende-se verificar esta propriedade pelo método da $\Delta^{\circlearrowleft}$-coindução. Tal como no Exemplo 4.1.10, o algoritmo da $\Delta^{\circlearrowleft}$-coindução reduz o par de termos*

```
(tail(odd(merge(s,s'))),tail(s))
```

ao par

```
(even(merge(s',tail(s))),tail(s)),
```

que vai agora ser adicionado como hipótese para a restante prova. Continuando a seguir o Exemplo 4.1.10, chega-se ao par

$$\texttt{(even(merge(tail(s'),tail(tail(s))),tail(tail(s))),}$$

que corresponde ao caso $(\delta[\theta(\texttt{c[t]})], \delta[\theta(\texttt{c[t']})])$*, para* $\texttt{c=z}_{stream}$*,* δ=`tail` *e* θ *tal que* θ`(s)=tail(s)`*, e portanto fica provado*

$$\texttt{STREAM}_{merge} \models_{\approx_{Obs}} \forall \texttt{s:stream.odd(merge(s,s')=s},$$

tal como se pretendia.

◇

O algoritmo que se segue está implementado na linguagem de verificação BOBJ, e foi inicialmente apresentado em [GLR00b]:

Algoritmo 3 (*Circular Coinductive Rewriting*)**.** *O algoritmo recebe como INPUT uma especificação flat* $SP = \langle(\Sigma, \Gamma), \Phi\rangle$*, uma cobase* Δ *e um par de* $Sig(SP)$*-termos* (t, t')*. Devolve a resposta* ***SIM*** *caso* $t \approx^{\Gamma}_{Obs} t'$*, e devolve* ***FALSO*** *ou não termina em caso contrário.*

1. *Sejam* $\mathcal{C} = \emptyset$ *e* $\mathcal{G} = \{(t, t')\}$*;*
2. *Para cada* (r, r') *de* $\mathcal{G}$ *fazer:*
   (a) *Passar* (r, r') *de* $\mathcal{G}$ *para* $\mathcal{C}$*;*
   (b) *Para todo* $\delta \in \Delta$:
      i. *Sejam* $u = bnf_{\mathcal{C}}(\delta[r, x])$ *e* $u' = bnf_{\mathcal{C}}(\delta[r', x])$*;*
      ii. *Se* $u \neq u'$ *então adicionar* (u, u') *a* $\mathcal{G}$*.*

onde $bnf_{\mathcal{C}}(r)$ *representa a derivação do termo* r *por reescrita* Γ*-observacional, segundo o sistema induzido pelas equações de* $\Phi \cup \bar{\mathcal{C}}$ *para* $\bar{\mathcal{C}} = \{c \approx c' | (c, c') \in \mathcal{C}\}$*, aplicando de seguida as equações de* $\mathcal{C}$ *a* s*, caso todas as operações de* s *(zero ou mais) estejam em* $\Gamma - \Delta$*.*

Coindução Circular Condicional com Análise de Casos

O mais recente algoritmo desta família é o *Conditional Circular Coinductive Rewriting with Case Analysis* apresentado em [GLR02]. Este algoritmo estende o *Circular*

Coinductive Rewriting à verificação de equações condicionais e introduz o conceito de *análise de casos* na verificação Γ-observacional de propriedades.

Intuitivamente, a verificação observacional de uma equação condicional numa determinada especificação por este algoritmo baseia-se no seguinte: tomam-se as premissas da equação condicional por hipótese (pela sua passagem a axiomas da especificação) e verifica-se, na especificação resultante, a sua conclusão por *Circular Coinductive Rewriting*. Este princípio apoia-se num importante resultado:

Teorema 4.1.15 (Teorema da dedução)**.** *Sejam* $SP = \langle(\Sigma, \Gamma), \Phi\rangle$ *uma especificação flat,* $t_1, \ldots, t_n, t'_1, \ldots, t'_n$ *termos de* Σ *sem variáveis,* $\Phi' = \Phi \cup \{(\forall \emptyset).t_1{=}t_2, \ldots, (\forall \emptyset).t_n{=}t'_n\}$ *e* $SP' = \langle(\Sigma, \Gamma), \Phi'\rangle$*. Tem-se que*

$$SP' \models (\forall X).t{=}t' \text{ sse } SP \models (\forall X).(t_1{=}t'_1) \wedge \cdots \wedge (t_n{=}t'_n) \rightarrow t{=}t'.$$

A demonstração do resultado pode ser encontrada em [Roş00, Proposição 26]. Contudo, esta passagem não pode ser directa, na medida em que, para a aplicação do Teorema anterior, é necessário eliminar os quantificadores universais das premissas da equação condicional. O problema é ultrapassado pela criação de uma nova constante na assinatura da especificação por cada variável destas equações via *"Teorema das Constantes"*:

Teorema 4.1.16 (Teorema das constantes)**.** *Sejam* $SP = \langle(\Sigma, \Gamma), \Phi\rangle$ *uma especificação flat, a* $(\Sigma \cup X)$*-equação condicional* $(\forall Y, X).(t_1{=}t'_1) \wedge \cdots \wedge (t_n{=}t'_n) \rightarrow t{=}t'$ *e a* Σ*-equação condicional* $(\forall Y).(t_1{=}t'_1) \wedge \cdots \wedge (t_n{=}t'_n) \rightarrow t{=}t'$ *para* $(\Sigma \cup X)$ *assinatura obtida de* Σ *pela introdução das variáveis de* X *como constantes. Tem-se que*

$$SP \models (\forall Y, X).(t_1{=}t'_1) \wedge \cdots \wedge (t_n{=}t'_n) \rightarrow t{=}t'$$

sse

$$SP' \models (\forall Y).(t_1{=}t'_1) \wedge \cdots \wedge (t_n{=}t'_n) \rightarrow t{=}t',$$

para $SP' = \langle((\Sigma \cup X), \Gamma), \Phi\rangle$.

A demonstração do resultado pode ser consultada, por exemplo, em [RG00, Teorema 10]. Desta forma, quando se pretende verificar observacionalmente uma equação condicional numa especificação, procede-se primeiro à eliminação dos quantificadores universais das premissas pela aplicação do Teorema 4.1.16, e depois, procede-se à verificação na nova especificação obtida pelo Teorema 4.1.15 (por *Circular Coinductive Rewriting*).

O uso de análise de casos na prática de demonstração é muito frequente. Por exemplo, é muito usual, quando se pretende verificar uma propriedade sobre os inteiros, proceder-se separadamente à verificação da propriedade nos casos $n = 0, n > 0$ e $n < 0$. O algoritmo *Conditional Circular Coinductive Rewriting with Case Analysis* encara a verificação observacional de uma equação $t = t'$ pela análise dos casos $\{c_i, \ldots, c_n\}$ como a conjunção das verificações $SP \models_{\approx_{Obs}} t{=}t'$ caso c_i, para todo o $1 \leq i \leq n$. A apresentação completa destes algoritmos pode ser encontrada em [GLR02].

4.2 Abordagem da Lógica Algébrica Abstracta à Igualdade Observacional

A abordagem à verificação observacional de software que se apresenta nesta Secção é essencialmente influenciada pelos trabalhos da *Lógica Algébrica Abstracta-LAA*. A *LAA* tem por objecto de estudo a associação de classes de álgebras a sistemas lógicos. Este trabalho é feito no sentido de unificar e generalizar a vários sistemas lógicos, resultados verificados em alguns sistemas lógicos e algumas classes de álgebras em particular, como são exemplo das ligações entre o cálculo proposicional e as álgebras Boolenas. Um marco central para a afirmação da *LAA* como área autónoma da lógica, foi a formalização do conceito de *sistema dedutivo algebrizável* de *Blok* e *Pigozzi* (cf. [BP98]).

Muitos dos assuntos trabalhados neste texto podem ser estudados segundo esta abordagem. Por exemplo, o raciocínio observacional em equações condicionais é apresentado em [MP07], um estudo sobre refinamentos (Secções 2.4 e 3.4) apresenta-se em [Mar06], a equivalência observacional entre álgebras (Secção 3.3) é estudado em [Mar08] e propriedades de fecho nas classes de modelos observacionais é feito em [Mar07].

Perante a abordagem da *LAA*, as especificações observacionais são vistas como *k-lógicas escondidas* e os seus modelos por *k-estruturas de dados observáveis*. O conceito de *k-lógica* é de grande abrangência, na medida em que capta várias lógicas conhecidas, como são os casos da lógica equacional (*2-lógica*) e das lógicas *booleanas*(*1-lógica*). Nesta abordagem, trabalham-se exclusivamente especificações observáveis, isto é, especificações descritas por axiomas de género observável. Apresentam-se nesta Secção alguns conceitos e resultados relevantes deste estudo aplicados ao caso equacional.

4.2.1 Lógica Equacional Escondida

Notação e Definições

Tal como se referiu, particularizam-se aqui alguns resultados estudados para as k-lógicas escondidas ao caso equacional. Apresentou-se na Secção 2.3 que uma especificação algébrica é constituída por uma assinatura e por uma classe de álgebras dessa assinatura denominada por classe de modelos da especificação. No contexto das lógicas equacionais escondidas existem, para esse efeito, as chamadas Σ-*estruturas de dados* que podem ser vistas como estruturas de primeira ordem sobre uma linguagem com um símbolo de relação de identidade:

Definição 4.2.1 (Estrutura de dados observável)**.** *Sejam $\Sigma = (S, \Omega)$ uma assinatura e $Obs \subseteq S$ um conjunto de géneros observáveis. Uma* Σ-estrutura de dados Observável*, é um par $\boldsymbol{A} = \langle A, F\rangle$, onde A é uma Σ-álgebra e $F \subseteq A_{Obs} \times A_{Obs}$ para $A_{Obs} = (A_s)_{s\in Obs}$.*

Numa estrutura $\mathbf{A} = \langle A, F\rangle$, F pode ser visto como a relação binária que interpreta a relação de igualdade em A, desempenhando o papel dos valores de verdade na estrutura. Nesta Secção não se quantificam variáveis e utilizam-se as notações $\psi(\bar{x}{:}\bar{Q})$ para denotar $\psi(x_1 : s_1, \ldots, x_n, s_n)$ e $\psi(\bar{b}) = \psi(b_1 : A_{s_1}, \ldots, b_n : A_{s_n})$.

Sintaxe

Na presente Secção considera-se o conjunto das fórmulas sobre uma assinatura Σ como sendo apenas o conjunto de equações e de equações condicionais sobre essa mesma assinatura. Denotando-se este conjunto por $Fm(\Sigma)$, por forma a distingui-lo do conjunto de fórmulas (de primeira-ordem) $Flas(\Sigma)$ da Definição 2.1.18.

Definição 4.2.2 (Relação de consequência)**.** *Sejam $\Sigma = (S, \Omega)$ uma assinatura heterogénea. A relação $\vdash \subseteq \mathcal{P}(Eq(X)_{Obs}) \times Eq(X)_{Obs}$ diz-se uma* relação de consequência *se para quaisquerΓ e Δ conjuntos de Σ-equações observáveis:*

1. *$\Gamma \vdash \psi$ para todo $\psi \in \Gamma$;*
2. *$\Gamma \vdash \psi$ e $\Gamma \subseteq \Delta$ implica que $\Delta \vdash \psi$;*
3. *Se $\Gamma \vdash \psi$, e $\Delta \vdash \gamma$ para todo $\gamma \in \Gamma$, então $\Delta \vdash \psi$.*

A relação de consequência $\vdash$ diz-se *finitária* quando $\Gamma \vdash \phi$ implica que $\Delta \vdash \phi$ para algum subconjunto Δ de Γ tal que $\bigcup_{s\in S} \Delta_s$ é finito; a relação diz-se *invariante por*

substituições se para toda a substituição $\sigma : Fm(\Sigma) \to Fm(\Sigma)$, se tem que $\Gamma \vdash \phi$ implica que $\sigma(\Gamma) \vdash \sigma(\phi)$. Uma função com estas características denomina-se por *substituição*.

Definição 4.2.3 (Lógica e Lógica Escondida)**.** *Sejam $\Sigma = (S, \Omega)$ uma assinatura heterogénea, $\vdash_{\mathfrak{L}}$ uma relação de consequência invariante por substituições entre Σ-fórmulas e $Obs \subseteq S$ um conjunto de géneros observáveis. Uma Σ-lógica escondida é um par $\mathfrak{L} = \langle \Sigma, \vdash_{\mathfrak{L}} \rangle$. Quando $\vdash_{\mathfrak{L}}$ é finitária diz-se que $\mathfrak{L}$ é* especificável.

Um importante conceito neste contexto é o de *teoria*. Uma teoria T é um conjunto de fórmulas fechado pela consequência lógica, isto é, T é teoria de $\mathfrak{L}$ se e só se $T \vdash_{\mathfrak{L}} \psi$ implica que $\psi \in T$. O conjunto de todas as teorias de $\mathfrak{L}$ denota-se por $Th(\mathfrak{L})$. A menor teoria que contém um conjunto de fórmulas Γ é denominada por conjunto das $\mathfrak{L}$-*consequências* de Γ e denota-se por $Cn_{\mathfrak{L}}(\Gamma)$. Uma fórmula derivável por $\mathfrak{L}$ do conjunto vazio de fórmulas ($\emptyset \vdash_{\mathfrak{L}} \psi$) denomina-se por *teorema* de $\mathfrak{L}$ ou $\mathfrak{L}$-*teorema*. O conjunto de todos os teoremas de uma lógica $\mathfrak{L}$ denota-se por $Thm(\mathfrak{L})$. Tal como foi referido, trabalha-se aqui essencialmente o caso equacional, onde as lógicas são definidas no estilo de *Hilbert* (a partir de um conjunto de axiomas e regras de inferência, na forma de equações e equações condicionais respectivamente), estando a relação de consequência lógica, definida de forma natural.

Definição 4.2.4 (Lógica Equacional Livre Escondida)**.** *Sejam $\Sigma = (S, \Omega)$ uma assinatura heterogénea, e $Obs \subseteq S$ um conjunto de géneros observáveis para Σ. A* lógica equacional escondida sobre Σ a respeito de *Obs, designa-se por HEL_{Σ}^{Obs} livre, é a lógica escondida definida da seguinte forma:*

Axiomas *$x : s{=}x : s$, para todo o género $s \in Obs$;*

Regras de inferência *Para todos os géneros $s, v \in Obs$:*

$x : s{=}y : s \to y : s{=}x : s$;

$x : s{=}y : s, y : s{=}z : s \to x : s{=}z : s$;

$\psi : s{=}\phi : s \to \zeta(x/\psi) : v{=}\zeta(x/\phi) : v$, *para toda* $\zeta \in T_{\Sigma}(X)_v$, $x \in X_s$.

Sempre que for claro no contexto, escreve-se HEL_{Σ} em vez de HEL_{Σ}^{Obs}.

As lógicas obtidas da lógica equacional livre pela introdução de novos axiomas (na forma de equações) e de novas regras de inferência (na forma de equações condicionais) são designadas por *HEL's aplicadas especificáveis*, ou simplesmente *HEL*

Exemplo 4.2.5 (FLAGS ([MP07]))**.** *Considere-se a assinatura* Σ_{FLAGS} *utilizada na especificação* `FLAGS` *apresentada no Exemplo 3.1.10. A lógica* $\mathfrak{L}_{FLAGS}$*, HEL aplicada sobre a assinatura* Σ_{FLAGS}*, é obtida de* HEL_{FLAGS}*, pela introdução dos axiomas de* `FLAGS`*. Observe-se que a axiomatização de* `FLAGS` *é observável, isto é, constituída exclusivamente por axiomas observáveis.*

◇

Exemplo 4.2.6 (STREAM (Adaptado de [MP07]))**.** *Considere-se a assinatura* Σ_{STREAM} *utilizada na especificação* `STREAM` *apresentada no Exemplo* 3.1.11*. A HEL aplicada* $\mathfrak{L}_{STREAM}$ *sobre* Σ_{STREAM}*, é a seguinte lógica* $\mathfrak{L}_{STREAM}$*:*

Axiomas extra-lógica `head(x::y)=x`*;*

`head(tail`$^{n+1}$`(x::y))=head(tail`^{n}`(y))`*, para todo o* $n \geq 0$*;*

`head(tail(push(empty)))=zero`*, para todo* $n \geq 0$*;*

Regras de inferência extra-lógica `s(x)=s(y)` $\rightarrow$ `x=y`.

Observe-se que a axiomatização apresentada difere da axiomatização (finita) que se apresentou no Exemplo 3.1.11. Esta diferença deve-se ao facto desta abordagem utilizar exclusivamente axiomatizações observáveis, isto é, sob a forma de axiomas observáveis.

◇

Uma outra classe de lógicas importante para o propósito do texto é a classe das *un-hidden equational logics-UHEL*. Esta classe define-se de forma análoga à de *HEL* (Definição 4.2.4), considerando todos os géneros como observáveis.

Semântica

Definição 4.2.7 (Consequência semântica)**.** *Seja* C *uma classe de* Σ*-estruturas de dados.*

1. *Uma igualdade* t=t' *do género observável* s' *diz-se uma* consequência válida *do conjunto de* Σ*-fórmulas observáveis* Γ *em* C*, em símbolos* $\Gamma \models_C t$=t'*, se:*

$$(\forall \langle A, F\rangle \in C)(\forall \alpha : X \rightarrow A)[(\forall (r{=}r') \in \Gamma), (I_\alpha(r), I_\alpha(r')) \in F_s)$$

$$\Rightarrow (I_\alpha(t), I_\alpha(t')) \in F_{s'}].$$

2. *Uma equação t=t′ do género observável s é válida em C se para toda a Σ-estrutura de dados* $\boldsymbol{A} = \langle A, F\rangle$ *em C, para toda a valoração* $\alpha : X \to A$*, se tem que* $(I_\alpha(t), I_\alpha(t')) \in F_s$.

3. *Uma regra* $\phi_1, \ldots, \phi_{n-1} \to \phi_n$ *é* válida em C *se* $\{\phi_1, \ldots, \phi_{n-1}\} \models_C \phi_n$

Caso $C = \{\mathbf{A}\}$, escreve-se $\Gamma \models_{\mathbf{A}} \psi$ em vez de $\Gamma \models_{\{\mathbf{A}\}} \psi$.

Definição 4.2.8 (Modelo de uma lógica escondida)**.** *Uma Σ-estrutura de dados* $\boldsymbol{A}$ *é modelo da HEL* $\mathfrak{L}$ *se qualquer* $\mathfrak{L}$*-consequência sintáctica for consequência semântica de* $\mathfrak{L}$*, isto é,*

$$\textit{se } \Gamma \vdash_{\mathfrak{L}} \psi \textit{ então } \Gamma \models_{\boldsymbol{A}} \psi.$$

Denota-se a classe de modelos de $\mathfrak{L}$ *por* $[\![\mathfrak{L}]\!]$.

Os modelos de uma HEL_Σ livre são Σ-estruturas de dados $\mathbf{A} = \langle A, F\rangle$ onde A representa uma Σ-álgebra e F uma congruência na parte observável de A. Em particular tem-se, o caso $\langle A, id_{A_{Obs}}\rangle$, onde $id_{A_{Obs}}$ representa a relação de identidade nos géneros observáveis.

Teorema 4.2.9 (Teorema da completude das lógicas escondidas)**.** *Seja* $\mathfrak{L}$ *uma lógica escondida. Tem-se que*

$$\vdash_{\mathfrak{L}} = \models_{[\![\mathfrak{L}]\!]},$$

ou seja, tem-se que para todo o conjunto de fórmulas Γ *e para qualquer fórmula* ψ*,* $\Gamma \vdash_{\mathfrak{L}} \psi$ *sse* $\Gamma \models_{[\![\mathfrak{L}]\!]} \psi$.

Este resultado aparece enunciado e demonstrado para o caso geral das k-lógicas escondidas em [Mar04, Teorema 2.3.18].

4.2.2 Igualdade Observacional como a Congruência de *Leibniz*

Apresenta-se nesta Secção a relação de igualdade observacional, como um caso particular da relação da congruência de *Leibniz*, entidade central do estudo da *LAA*.

Dada uma Σ-estrutura de dados $\mathbf{A} = \langle A, F\rangle$ e uma relação de congruência em A Θ, diz-se que a relação Θ é (observacionalmente) compatível com F se, para quaisquer $a_1, a_1', a_2, a_2' \in A_s$, $s \in Obs$ se tem que

$$\text{se } (a_1, a_1') \in \Theta_s \text{ e } (a_2, a_2') \in \Theta_s \text{ então } (a_1, a_2) \in F_s \text{ sse } (a_1', a_2') \in F_s,$$

e portanto, F_s é o seguinte *produto cartesiano* de classes:

$$F_s = \bigcup_{\langle a,b\rangle \in F_s} (a/\Theta_s) \times (b/\Theta_s).$$

A definição que se segue é baseada no seguinte critério de *Leibniz*:

"Dois objectos no universo de discurso são iguais se partilham todas as propriedades que podem ser expressas nessa linguagem de discurso."

Definição 4.2.10 (Congruência de *Leibniz*)**.** *Seja $\boldsymbol{A} = \langle A, F\rangle$ uma Σ-estrutura de dados. A maior relação de congruência em A compatível com F denomina-se por* congruência de Leibniz*, e denota-se por $\Omega_A(F)$.*

Prova-se que esta congruência existe sempre (cf. [Mar04, Secção de 2.2.3]). Uma vez que a congruência $\Omega(F)$ é maximal (Definição 4.2.10), é possível adaptar, de forma directa, o método de coindução apresentado no Algoritmo 1 para a relação $\approx^A_{Obs}$ à relação $\Omega_A(F)$:

Algoritmo 4 (Método da coindução generalizada)**.** *Seja $\boldsymbol{A} = \langle A, F\rangle$ uma estrutura. Para provar que o par $\langle a, a'\rangle \in A_s$ está em $\Omega(F)_s$:*

1. *Definir uma relação R em A;*
2. *Mostrar que R é uma congruência em $\boldsymbol{A}$ compatível com F;*
3. *Mostrar que $\langle a, a'\rangle \in R$.*

Efectivamente, tal como se vê pelo Corolário 4.2.13, quando se considera $F = id_{A_{Obs}}$ obtém-se o próprio Algoritmo 1, sendo portanto o Algoritmo 4 uma generalização do Algoritmo 1. Por esta razão é aqui designado por *método da coindução generalizada*.

Teorema 4.2.11. [MP07, Teorema 2.16] *Sejam Σ uma assinatura e $\boldsymbol{A} = \langle A, F\rangle$ uma estrutura sobre Σ. São equivalentes as seguintes afirmações:*

- $(a, a') \in \Omega_A(F)$*;*
- *Para todos os pares (ψ, ϕ) tais que $\psi, \phi \in \mathcal{C}^{Obs}_\Sigma$:*

$$(\psi^A(a, \bar{b}), \phi^A(a, \bar{b})) \in F \text{ sse } (\psi^A(a', \bar{b}), \phi^A(a', \bar{b})) \in F.$$

A caracterização da congruência de *Leibniz* via contextos observáveis apresentada no Teorema que se segue, mostra que esta congruência pode ser vista como uma generalização do conceito de $\approx^A_{Obs}$, o que se comprova no Corolário 4.2.13.

Teorema 4.2.12. [MP07, Teorema 2.23] *Sejam $\Sigma = (S, \Omega)$ uma assinatura com $Obs \subseteq S$ conjunto de géneros observáveis e $\boldsymbol{A} = \langle A, F \rangle$ um modelo de HEL_Σ. Para todo $s \in S$, e para todos $a, a' \in A_s$, tem-se que $a \approx_{\Omega(F)_s} a'$ sse para todo o contexto observável $\psi(z_s, x_1, \ldots, x_n)$ do género s,*

$$(\psi^A(a, b_1, \ldots, b_n), \psi^A(a', b_1, \ldots, b_n)) \in F_s,$$

para quaisquer $b_i \in A_{s_i}$, para todo $1 \leq i \leq n$.

O resultado que se segue, estabelece a ponte entre a lógica algébrica e a especificação e verificação formal de sistemas com dados encapsulados:

Corolário 4.2.13. *Sejam a, a' elementos de A_s. Tem-se que*

$$a \approx^A_{Obs} a' \text{ sse } (a, a') \in \Omega_A(id_{A_{Obs}}).$$

Para terminar apresenta-se mais uma caracterização desta relação:

Corolário 4.2.14. *$\Omega(F)$ é a maior congruência tal que $(\Omega(F))_{Obs} = F$.*

4.2.3 Especificação Observacional

Definição 4.2.15 (Consequência semântica observacional)**.** *Sejam $\Sigma = (S, \Omega)$ uma assinatura, $Obs \subseteq S$ um conjunto de géneros observáveis para Σ e C uma classe de Σ-estruturas de dados.*

1. *Uma igualdade $t{=}t'$ diz-se uma* consequência observacional válida *do conjunto de Σ-igualdades Γ, em símbolos $\Gamma \models^{Obs}_C t{=}t'$, quando se tem que: para toda $\boldsymbol{A} = \langle A, F \rangle \in C$, para toda a valoração $\alpha : X \to A$, se $I_\alpha(s) \approx^A_{Obs} I_\alpha(s')$ para toda a igualdade $s{=}s' \in \Gamma$ então $I_\alpha(t) \approx^A_{Obs} I_\alpha(t')$.*

2. *Uma equação $t{=}t'$ diz-se* observacionalmente válida *em C se $\models^{Obs}_C t{=}t'$. A equação condicional $t_1{=}t'_1 \wedge \cdots \wedge t_n{=}t'_n \to t{=}t'$ diz-se* observacionalmente válida *em C se $\{t_1{=}t'_1, \ldots, t_n{=}t'_n\} \models^{Obs}_C t{=}t'$.*

Tal como já foi referido, a introdução do predicado da igualdade observacional transporta para a lógica standard algumas das características dos sistemas de transição de estados, apontados como os sistemas mais adequados para o estudo semântico de programas com dados encapsulados. Assim, é importante saber quando é que a

consequência observacional de uma determinada HEL aplicada $\mathfrak{L}$, é axiomatizada no sentido standard, isto é, quando existe uma lógica $UHEL$ $\mathfrak{L}'$ tal que todo teorema observacional em $\mathfrak{L}$ seja teorema em sentido estrito em $\mathfrak{L}'$:

Definição 4.2.16 (Lógica Observacionalmente especificável)**.** *Seja $\mathfrak{L}$ uma HEL. Diz-se que a lógica $\mathfrak{L}$ é* observacionalmente especificável *se existe uma $UHEL$ $\mathfrak{L}'$ sobre a mesma assinatura, tal que $\models^{Obs}_{\mathfrak{L}} = \vdash_{\mathfrak{L}'}$, isto é, tal que para qualquer conjunto de equações $E \cup \{t{=}t'\}$,*

$$E \models^{Obs}_{\mathfrak{L}} t{=}t \text{ sse } E \vdash_{\mathfrak{L}'} t{=}t'.$$

A lógica $\mathfrak{L}'$ denomina-se por especificação observacional *de $\mathfrak{L}$.*

O *sistema de equivalência* é uma entidade muito utilizada no estudo de propriedades de sistemas dedutivos. Vários resultados acerca da observabilidade das HEL foram obtidos pelo estudo de propriedades destes sistemas, entre eles a caracterização da especificabilidade da consequência observacional (Teorema 4.2.20) e um método de verificação observacional (Teorema 4.2.21). Estes resultados são apresentados em [MP07], ou em [Mar04], para o caso geral das k-lógicas escondidas.

Define-se, de seguida, o conceito de *sistema de equivalência*:

Definição 4.2.17 (Sistema de equivalência)**.** *Seja $E := ((E_{s,v}(x,y)_{v \in Obs})_{s \in S}$ uma S-família tal que para todo o $s \in S$, $E_s(x,y)$ represente um conjunto possivelmente infinito de equações observáveis $\psi(x{:}s, y{:}s){:}v{=}\phi(x{:}s, y{:}s){:}v$. Então, E diz-se um* sistema de equivalência para a HEL $\mathfrak{L}$ *se, para todos os géneros $s \in Obs$, $E_s(x{:}s, y{:}s) = \{x{:}s{=}y{:}s\}$ e para todo o $s \in S \setminus Obs$, se verificam as seguintes condições:*

1. $\vdash_{\mathfrak{L}} E_s(x{:}s, x{:}s)$;
2. $E_s(x{:}s, y{:}s) \vdash_{\mathfrak{L}} E_s(y{:}s, x{:}s)$;
3. $\{E_s(x{:}s, y{:}s), E_s(y{:}s, z{:}s)\} \vdash_{\mathfrak{L}} E_s(x{:}s, z{:}s)$;
4. *Para toda o símbolo de operação $f : s_1, \ldots, s_n \to s \in \Sigma$,*

   Caso $s \in S \setminus Obs$ então:

   $\bigcup\{E_{s_i}(x_i : s_i, y_i : s_i) | s_i \in S \setminus Obs, i \leq n\} \cup \{x_i{=}y_i | s_i \in Obs, i \leq n\}$

   $\vdash_{\mathfrak{L}} E_s(f(x_1, \ldots, x_n){:}s, f(y_1, \ldots, y_n){:}s)$

Caso $s \in Obs$

$$\bigcup\{E_{s_i}(x_i{:}s_i, y_i{:}s_i) | s_i \in S \setminus Obs, i \leq n\} \cup \{x_i{=}y_i | s_i \in Obs, i \leq n\}$$
$$\vdash_{\mathfrak{L}} f(x_1, \ldots, x_n){:}s{=}f(y_1, \ldots, y_n){:}s$$

Uma HEL $\mathfrak{L}$ diz-se *equivalencial* se possui um sistema de equivalência E. No caso de E, sistema de equivalência de $\mathfrak{L}$, ser tal que $\bigcup_{s \in Obs} E_{s,h}$ seja finito para todo o género $h \in S \setminus Obs$, então diz-se que $\mathfrak{L}$ é *finitamente equivalencial*.

Apresentam-se de seguida dois exemplos de sistemas de equivalência:

Exemplo 4.2.18 (FLAGS ([MP07])). *Considere-se a lógica* $\mathfrak{L}_{FLAGS}$*, definida no Exemplo 4.2.5. O sistema* $E = \langle E_{bool}, E_{flag} \rangle$ *tal que* E_{bool}(`x:bool,y:bool`) $= \{$`x=y`$\}$*, e* E_{flag}(`x:flag,y:flag`) $= \{$`?up(x)=?up(y)`$\}$ *é sistema equivalencial de* $\mathfrak{L}_{FLAGS}$*.*

◇

Exemplo 4.2.19 (STREAM (Adaptado de [MP07])). *Considere-se a lógica* $\mathfrak{L}_{STREAM}$*, definida no Exemplo 4.2.6. O sistema* $E = \langle E_{nat}, E_{stream} \rangle$ *tal que* E_{nat}(`x:nat,y:nat`) $=$ $\{$`x=y`$\}$*, e* E_{nat}(`x:stream,y:stream`) $= \{$`head(tail`$^n(x)) =$ `head(tail`^{n}`(y))` $| n \geq 0\}$ *é sistema equivalencial (não finito) de* $\mathfrak{L}_{STREAM}$*.*

◇

Tal como se discute no Capítulo 5, existem HEL's não observacionalmente especificáveis, sendo por esta razão importante caracterizar uma lógica quanto à sua especificabilidade observacional. O Teorema que se segue estabelece uma caracterização desta propriedade:

Teorema 4.2.20. [MP07, Teorema 3.20] *Uma* HEL $\mathfrak{L}$ *especificável é observacionalmente especificável sse é finitamente equivalencial.*

Segue-se um Teorema que estabelece um método efectivo de prova de teoremas observacionais. Também aqui o conceito de sistema de equivalência toma o papel central:

Teorema 4.2.21. [MP07, Teorema 3.22] *Seja* $\mathfrak{L}$ *uma* HEL *equivalencial com sistema de equivalência* E*. Então, a equação condicional*

$$t_1{=}t'_1, \ldots, t_n{=}t'_n \rightarrow t{=}t',$$

tal que $t_i, t'_i \in s_i$*,* $t, t' \in s$ *é observacionalmente válida em* $\mathfrak{L}$ *sse*

$$\bigcup_{i \leq n} E_{s_i}(t_i : s_i, t'_i : s_i) \vdash_{\mathfrak{L}} E_s(t : s, t' : s).$$

Observe-se que o método de verificação de propriedades observacionais introduzido pelo Teorema anterior é muito semelhante ao método da Δ-coindução apresentado na Secção 4.1.2 (Algoritmo 2). De facto, um sistema de equivalência E de uma HEL $\mathfrak{L}$ pode ser visto como um caso específico de uma cobase (Definição 4.1.7), na qual só se admitem equações observáveis. No exemplo que se segue, recorre-se ao Teorema 4.2.21, para verificar uma propriedade observacional:

Exemplo 4.2.22 (FLAGS ([MP07])). *Provou-se nos Exemplos 3.5.13 e 4.1.4 que* `rev(rev(F))` $\approx_{Obs}$ `F`. *Pretende-se agora provar a mesma propriedade com recurso ao Teorema 4.2.21. Considere-se para tal, a* $HEL_{\Sigma_{FLAGS}}$ *e o sistema de equivalência* E_{FLAGS} *apresentados nos Exemplos 4.2.5 e 4.2.18. Tem-se que:*

$\vdash_{\mathfrak{L}_{\Sigma_{FLAGS}}}$ `?up(rev(rev(F)))=?up(rev(rev(F)))`	E_{FLAGS} *sistema de equivalência*
$\vdash_{\mathfrak{L}_{\Sigma_{FLAGS}}}$ `?up(rev(rev(F)))=¬?up(rev(F))`	*axioma (3) de* `FLAG`
$\vdash_{\mathfrak{L}_{\Sigma_{FLAGS}}}$ `?up(rev(rev(F)))=¬¬?up(F)`	*axioma (3) de* `FLAG`
$\vdash_{\mathfrak{L}_{\Sigma_{FLAGS}}}$ `?up(rev(rev(F)))=?up(F)`	`¬¬x=x` *axioma de* `BOOL`

e portanto, sai do Teorema 4.2.21 que `FLAGS` $\models^{Obs}_{\mathfrak{L}}$ `rev(rev(F)) = F`, *tal como se pretendia.*

$\diamond$

A partir do estudo dos sistemas de equivalência, chegou-se ainda a outros resultados importantes relativamente a este assunto. Por exemplo, em [Mar07] estabelece-se uma hierarquia de várias classes de lógicas escondidas baseada no estudo das propriedades destes sistemas.

Capítulo 5

Incompletude das Lógicas Comportamentais

Nos Capítulos anteriores apresentaram-se alguns métodos de verificação de propriedades comportamentais em especificações algébricas, todos eles, de alguma forma, dependentes das axiomatização da igualdade comportamental em questão. Foi também por várias vezes abordada a problemática da procura de axiomatizações (finitárias) completas para este raciocínio, ficando subentendida a existência de especificações com igualdades comportamentais, não finitamente axiomatizáveis (na própria assinatura). Por exemplo, na Secção 3.5.1, por forma a contornar a infinitaridade da axiomatização "natural"da igualdade observacional numa especificação (ver Exemplo 3.5.8), apresentou-se um método a partir do qual se constrói uma nova especificação a partir da extensão da primeira pela introdução de novos géneros, operações e axiomas, na qual se axiomatiza, de forma finita, a igualdade observacional em questão. Note-se que, apesar de se conseguir desta forma uma axiomatização finitária da igualdade observacional, a assinatura da nova especificação já não é a mesma e, por conseguinte, não implica que esta igualdade seja finitamente axiomatizável na assinatura da especificação inicial. Também na Secção 4.2.3 se aborda este assunto, aquando da apresentação de uma caracterização para a existência de axiomatizações finitárias da igualdade observacional em especificações *flat* (no contexto, em lógicas equacionais escondidas). A inexistência deste tipo de axiomatizações foi estudada em vários trabalho. Por exemplo, em [Wol87, Teorema 6.3], mostra-se a inexistência de axiomatizações finitárias (na forma de conjuntos finito de equações condicionais) para a igualdade observacional na

especificação `STREAM` (Exemplo 3.1.11) e em [Sch92], estuda-se esta inexistência relativa à especificação de um *contador encapsulado* acessível via atributo *"teste zero"*. Em [Mar07], estuda-se uma caracterização de especificações algébricas, quanto à natureza da axiomatização da igualdade observacional. Esta caracterização é feita com base em propriedades dos sistemas de equivalência (Definição 4.2.17) e faz a distinção entre as especificações que admitem *sistemas de equivalência parametrizados*, *sistemas de equivalência parametrizados finitos*, *sistemas de equivalência* e *sistemas de equivalência finitos*, complementando a caracterização de *Bidoit* e *Hennicker* (Secção 3.5.1), que apenas distingue igualdades finitamente e infinitamente axiomatizáveis.

Observe-se que, caso existissem sempre axiomatizações finitárias para todas as igualdades comportamentais, seria possível provar algoritmicamente todas as propriedades comportamentais em qualquer especificação, na medida em que se podia desenvolver um algoritmo que gerasse e verificasse todas as provas possíveis, nessas especificações. No entanto, a existência de igualdades comportamentais que não admitem axiomatizações deste tipo, levanta naturalmente a questão da existência ou não de algoritmos que, dada uma qualquer especificação e uma qualquer igualdade comportamental $\approx$, provem todas as propriedades $\approx$-comportamentais.

De facto, demonstra-se que o problema da satisfação comportamental não é recursivamente enumerável e, por conseguinte, que não existe nenhum algoritmo que, a partir de uma qualquer especificação e de uma fórmula como input, devolva a resposta "YES"quando a fórmula é comportamentalmente satisfeita ou que corra indefinidamente no caso contrário. Para provar este resultado, originalmente apresentado em [BR00], define-se uma especificação flat finita, onde o problema da satisfação observacional é Π^0_2-Hard e, por conseguinte, não recursivamente enumerável. O facto de o problema da satisfação comportamental ser Π^0_2-Hard não implica só a não existência de uma axiomatização completa, mas também o facto de não haver forma algorítmica de refutar proposições que não sejam consequência comportamental de determinada especificação. A este tipo de problemas chama-se problemas não co-recursivamente enumeráveis.

A técnica utilizada na demonstração é a de reduzir a verificação da satisfação observacional numa especificação observacional finita a um problema Π^0_2-Hard. O problema utilizado é o clássico problema da *Totalidade* (ver Secção 5.2.1). Apresenta-se, neste Capítulo (Secção 5.2.2), o resultado central de [BR00]. Numa primeira fase do

Capítulo, introduzem-se de forma sucinta alguns conceitos da teoria da computabilidade necessários ao acompanhamento do texto, nomeadamente, o de *função parcialmente recursiva* (Secção 5.1.1), as *máquinas de Turing* (Secção 5.1.4) e a *hierarquia aritmética de conjuntos* (Secção 5.1.3).

5.1 Teoria da Recursão e Computabilidade

Faz-se nesta Secção, de forma breve, a apresentação dos conceitos da *Teoria da Computação* e da *Teoria da Recursividade* necessários para o desenvolvimento do texto, nomeadamente a formalização do conceito de função *computável* ou *calculável* por um *algoritmo*, isto é, por um *processo efectivo*. São várias as formalizações propostas na literatura para este feito, destacando-se, de entre elas, os *algoritmos de Markov*, os *sistemas de derivação de Post*, o λ-*calculus de Church*, as *máquinas de Turing*, as *máquinas URM*, entre outros. Estes sistemas são também conhecidos por *modelos de computação*. Um marco central da Teoria da Computação é o facto de todas estas caracterizações, se mostrarem equivalentes, no sentido em que definem a mesma classe de funções - *Postulado de Church*. Esta classe de funções é denominada por *classe das funções recursivas*. Define-se nesta Secção esta classe de funções (Secção 5.1.1), assim como o modelo de computação das máquinas de Turing (Secção 5.1.4). Termina-se com a apresentação dos conceitos de *conjunto recursivo* e *recursivamente enumerável* (Secção 5.1.2) e *hierarquia aritmética de conjuntos* (Secção 5.1.3).

5.1.1 λ-Linguagem e Funções Parcialmente Recursivas

Define-se nesta Secção a classe das *funções parcialmente recursivas*. Para tal, define-se de seguida a linguagem λ. As duas construções básicas da linguagem λ são a aplicação e a abstracção, sendo que, a expressão $(t_1 t_2)$ denota a aplicação de t_1 a t_2 e, dada uma λ-expressão M, $\lambda x.M$ denota a função $x \to M(x)$. A regra que se segue relaciona estas duas construções:

Definição 5.1.1. *A seguinte regra denomina-se por* β-redução*:*

$$(\lambda n.M)N = [N/n]M,$$

onde $[N/n]M$ *resulta da substituição de* n *por* N *em* M*.*

A definição formal da linguagem λ:

Definição 5.1.2. *O λ vocabulário, considerando fixado o conjunto de constantes naturais C, é composto por:*

- *Um conjunto numerável X, cujos elementos se denominam por variáveis naturais;*
- *O símbolo λ denominado abstracção;*
- *Os símbolos (e) denominados parêntesis esquerdo e direito respectivamente.*

Definição 5.1.3. *O conjunto dos termos da λ-linguagem denota-se por Λ e define-se da seguinte forma (considerado fixado o conjunto de constantes naturais C):*

- $(c_1, \ldots, c_n) \in \Lambda$, *se* $c_1, \ldots, c_n \in C, n \in \mathbb{N}$;
- $x \in \Lambda$, *qualquer que seja o* $x \in X$;
- $(t_1 t_2) \in \Lambda$, *se* $t_1, t_2 \in \Lambda$;
- $(\lambda x_1 \cdots x_n.t) \in \Lambda$ *se* $x_1, \ldots, x_n \in X, t \in \Lambda$.

Exemplo 5.1.4. *São exemplos de λ-termos as seguintes expressões:*

$$v$$

$$(vv')$$

$$(\lambda v(vv'))$$

$$((\lambda v(vv'))v'')$$

$\diamond$

Observe-se que por exemplo o termo $(\lambda xy(M))$ representa uma função $f : \mathbb{N}_0^2 \to \mathbb{N}_0$, enquanto que o termo $(\lambda x(\lambda y(M)))$representa uma função $g : \mathbb{N}_0 \to [\mathbb{N}_0 \to \mathbb{N}_0]$.

Introduzem-se de seguida algumas operações de fecho no espaço das funções, nomeadamente as operações de composição, recursão e minimização.

Definição 5.1.5. *Sejam*

$$f = \lambda y_1 \cdots y_k.f(y_1, \ldots, y_k) \in [\mathbb{N}_0^k \to \mathbb{N}_0],$$

$$g_i = \lambda x_1 \cdots x_n.g_i(x_1, \ldots, x_n) \in [\mathbb{N}_0^n \to \mathbb{N}_0], 1 \leq i \leq k$$

funções. A função $h = \lambda x_1 \cdots x_n.h(x_1, \ldots, x_n) \in [\mathbb{N}_0^n \to \mathbb{N}_0]$ *tal que*

$$h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$$

diz-se definida por composição a partir de f, $g_1, \ldots, g_k$.

Definição 5.1.6. *Sejam*

$$f = \lambda x_1 \cdots x_n.f(x_1, \ldots, x_n) \in [\mathbb{N}_0^n \to \mathbb{N}_0],$$

$$g = \lambda x_1 \cdots x_n yz.g(x_1, \ldots, x_n, y, z) \in [\mathbb{N}_0^{n+2} \to \mathbb{N}_0]$$

funções. A função $h = \lambda x_1 \cdots x_n y.h(x_1, \ldots, x_n, y) \in [\mathbb{N}_0^{n+1} \to \mathbb{N}_0]$ *tal que*

$$h(x_1, \ldots, x_n, 0) = (\lambda x_1 \cdots x_n y.h(x_1, \ldots, x_n, y))(x_1, \ldots, x_n, 0) =$$

$$= (\lambda x_1 \cdots x_n.f(x_1, \ldots, x_n))(x_1, \ldots, x_n) = f(x_1, \ldots, x_n))$$

e

$$h(x_1, \ldots, x_n, y+1) = (\lambda x_1 \cdots x_n y.h(x_1, \ldots, x_n, y))(x_1, \ldots, x_n, y+1) =$$

$$(\lambda x_1 \cdots x_n yz.g(x_1, \ldots, x_n, y, z))(x_1, \ldots, x_n, y, h(x_1, \ldots, x_n, y)) =$$

$$g(x_1, \ldots, x_n, y, h(x_1, \ldots, x_n, y))$$

diz-se função definida por recursão a partir de f *e de* g.

Definição 5.1.7. *Seja*

$$f = \lambda x_1 \cdots x_n y.f(x_1, \ldots, x_n, y) \in [\mathbb{N}_0^{n+1} \to \mathbb{N}_0].$$

A função $g = \lambda x_1 \cdots x_n.g(x_1, \ldots, x_n)$ *definida por* $g(x_1, \ldots, x_n) =$ *menor* y *tal que*

$$\begin{array}{l} f(x_1, \ldots, x_n, z) \text{ é definida para } z \leq y \\ f(x_1, \ldots, x_n, y) \text{ se existe } y \text{ nestas condições} \end{array} \tag{5.1}$$

diz-se definida por minimização a partir de f.

Normalmente funções deste tipo denotam-se por

$$g = \lambda x_1 .. x_n \mu y(f(x_1, \ldots, x_n, y) = 0) \in [\mathbb{N}_0^n \to \mathbb{N}_0].$$

Apresenta-se agora um sistema axiomático que define a classe das funções parcialmente recursivas:

Definição 5.1.8 (Função parcialmente recursiva [Ser93]). *O sistema lógico composto pelos axiomas:*

A.1 *A função* $\lambda x.0 \in [\mathbb{N}_0 \to \mathbb{N}_0]$ *é parcialmente recursiva;*

A.2 *A função* $\lambda x.x + 1 \in [\mathbb{N}_0 \to \mathbb{N}_0]$ *é parcialmente recursiva;*

A.3 *A função* $\lambda x_1 \cdots x_n.x_i \in [\mathbb{N}_0^n \to \mathbb{N}_0]$ *é parcialmente recursiva;*

e pelas regras de inferência:

Com *Sejam*

$$f = \lambda y_1 \cdots y_k.f(y_1, \ldots, y_k) \in [\mathbb{N}_0^k \to \mathbb{N}_0],$$

$$g_i = \lambda x_1 \cdots x_n.g_i(x_1, \ldots, x_n) \in [\mathbb{N}_0^n \to \mathbb{N}_0], 1 \leq i \leq k$$

funções parcialmente recursivas. Então, a função $h = \lambda x_1 \cdots x_n.h(x_1, \ldots, x_n) \in [\mathbb{N}_0^n \to \mathbb{N}_0]$ *tal que*

$$h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$$

é também parcialmente recursiva.

Rec *Sejam*

$$f = \lambda x_1 \cdots x_n.f(x_1, \ldots, x_n) \in [\mathbb{N}_0^n \to \mathbb{N}_0],$$

$$g = \lambda x_1 \cdots x_n yz.g(x_1, \ldots, x_n, y, z) \in [\mathbb{N}_0^{n+2} \to \mathbb{N}_0]$$

funções parcialmente recursivas. Então, a função $h = \lambda x_1 \cdots x_n y.h(x_1, \ldots, x_n, y) \in [\mathbb{N}_0^{n+1} \to \mathbb{N}_0]$ *tal que*

$$h(x_1, \ldots, x_n, 0) = (\lambda x_1 \cdots x_n y.h(x_1, \ldots, x_n, y))(x_1, \ldots, x_n, 0) =$$

$$= (\lambda x_1 \cdots x_n.f(x_1, \ldots, x_n))(x_1, \ldots, x_n) = f(x_1, \ldots, x_n))$$

e

$$h(x_1, \ldots, x_n, y + 1) = (\lambda x_1 \cdots x_n y.h(x_1, \ldots, x_n, y))(x_1, \ldots, x_n, y + 1) =$$

$$(\lambda x_1 \cdots x_n yz.g(x_1, \ldots, x_n, y, z))(x_1, \ldots, x_n, y, h(x_1, \ldots, x_n, y)) =$$

$$g(x_1, \ldots, x_n, y, h(x_1, \ldots, x_n, y))$$

é parcialmente recursiva.

Min *Seja*

$$f = \lambda x_1 \cdots x_n y.f(x_1, \ldots, x_n, y) \in [\mathbb{N}_0^{n+1} \to \mathbb{N}_0].$$

uma função parcialmente recursiva A função $g = \lambda x_1 \cdots x_n.g(x_1, \ldots, x_n)$ *definida por* $g(x_1, \ldots, x_n) =$ *menor y tal que*

$$\begin{array}{l} f(x_1, \ldots, x_n, z) \text{ é definida para } z \leq y \\ f(x_1, \ldots, x_n, y) \text{ se existe } y \text{ nestas condições} \end{array} \tag{5.2}$$

também é parcialmente recursiva.

denomina-se sistema **FPR**.

As funções mencionadas nos axiomas de **FPR** denominam-se de *funções iniciais*. Uma outra classe objecto de estudo da teoria da recursividade é a classe das funções primitivas recursivas. Esta classe é obtida pelo sistema anterior quando se retira a regra de inferência **Min**. Desta forma, tem-se que as funções parcialmente recursivas são todos os teoremas de **FPR**, i.e. a função f é parcialmente recursiva *sse* $\vdash_{\mathbf{FPR}} f$ e, portanto, verificar se uma função f é parcialmente recursiva não é mais do que fazer uma demonstração de f em **FPR** (cf. [Ser93]).

Dada uma função parcial $f : \mathbb{N}^n \to \mathbb{N}$, escreve-se $f(a_1, ..., a_n) \downarrow$ quando a função f está definida para os valores $(a_1, \ldots, a_n)$, isto é, quando existe um $b \in \mathbb{N}$ tal que $f(a_1, \ldots, a_n) = b$.

5.1.2 Conjuntos Recursivamente Enumeráveis e Reducibilidade

Conjuntos Recursivamente Enumeráveis

Um outro ponto de vista relativo à computabilidade é feito a partir da caracterização dos conjuntos recursivos e recursivamente enumeráveis. De forma intuitiva, um conjunto A diz-se recursivo se existe um processo algorítmico para decidir se um determinado elemento x pertence ou não a esse conjunto e diz-se *recursivamente enumerável* se existe um procedimento efectivo para listar os seus elementos. Formalmente:

Definição 5.1.9 (Função característica de um conjunto)**.** *Seja* $A \subseteq \mathbb{N}$. *Define-se* função característica do conjunto A *como a seguinte função:*

$$\chi_A(x) = \begin{cases} 1 \text{ caso } x \in A \\ 0 \text{ caso } x \in \bar{A} \end{cases}$$

Definição 5.1.10. *Um conjunto $A \subseteq \mathbb{N}$ diz-se* recursivo *se a sua função característica é recursiva e diz-se* recursivamente enumerável *se existe um função recursiva cujo contradomínio seja A.*

Seguem-se duas importantes caracterizações relativas à recursividade de conjuntos:

Teorema 5.1.11. [HR87, Secção 5.1 Teorema II] *Seja $A \subseteq \mathbb{N}$ um conjunto de números naturais. Tem-se que:*

A é recursivo sse A e $\bar{A}$ são recursivamente enumeráveis.

Teorema 5.1.12. [HR87, Secção 5.1 Teorema V] *O conjunto A é recursivamente enumerável sse é domínio de uma função parcialmente recursiva.*

Um importante conceito no que respeita o estudo da complexidade de problemas é o conceito de redutibilidade:

Definição 5.1.13. *O conjunto A* reduz-se *ao conjunto B se existe uma função recursiva f tal que*

$$(\forall x).[x \in A \Rightarrow f(x) \in B].$$

Quando f é injectiva escreve-se $A \leq_1 B$, e $A \leq_m B$ caso contrário (o m vem do termo em inglês many-one*, e o 1 é sugerido pelo termo* one-one reducible*). Escreve-se $A \equiv_1 B$ quando $A \leq_1 B$ e $B \leq_1 A$, e $A \equiv_m B$ quando $A \leq_m B$ e $B \leq_m A$.*

Um conjunto A diz-se completo segundo a ordem $\leq$ se A for recursivamente enumerável e para todo conjunto B tal que B recursivo, se tenha que $B \leq A$.

5.1.3 Hierarquia Aritmética

Uma importante ferramenta para a classificação de conjunto de naturais é a hierarquia aritmética. Os primeiros estudos sobre esta hierarquia foram apresentados em 1923 por *S. Kleene*, razão pela qual também é conhecida por *hierarquia de Kleene*.

Observe-se que se pode associar a cada conjunto $A \subseteq \mathbb{N}^k$ uma relação $R(a_1, \ldots, a_n)$, tal que: $A = \{(a_1, \ldots, a_n) \in \mathbb{N}^k | R(a_1, \ldots, a_n)\}$ e, por conseguinte, é possível classificar conjuntos de naturais a partir do estudo destas relações. No entanto, tal como foi referido, a hierarquia aritmética é utilizada para classificar conjuntos de naturais sendo o conjunto apresentado um conjunto de k-uplos de naturais. Efectivamente, este conjunto pode ser reduzido a um conjunto de naturais, caso se defina uma codificação

de k-uplos adequada, isto é, o conjunto $\{(a_1, \ldots, a_n) \in \mathbb{N}^k | R(a_1, \ldots, a_n)\}$ pode ser visto como o conjunto $\{a \in \mathbb{N} | R((a)_1, \ldots, (a)_n)\}$, onde $(a)_i$ representa a i-ésima componente obtida pela descodificação de a (cf. [DSW94]).

Definição 5.1.14 (Função característica de uma relação)**.** *Seja $R \subseteq \mathbb{N}^n$ uma relação k-ária. A função:*

$$\chi_R(x) = \begin{cases} 1 \text{ se } R(x_1, \ldots, x_n) \text{ é verdade} \\ 0 \text{ se } R(x_1, \ldots, x_n) \text{ não é verdade} \end{cases}$$

denomina-se por função característica de R.

Uma relação k-ária diz-se recursiva quando a sua função característica o for e diz-se recursivamente enumerável quando for domínio de uma função parcial recursiva. Dada uma relação $R(x_1, \ldots, x_n)$, a relação $S = \{(x_1, \ldots, x_{n-1}) | (\exists x_n) R(x_1, \ldots, x_n)\}$ diz-se uma *projecção* de R, e $S' = \{(x_1, \ldots, x_n) | \neg R(x_1, \ldots, x_n)\}$ diz-se *relação complementar* de R.

Definição 5.1.15 (Hierarquia aritmética)**.** *Uma relação R está na* hierarquia aritmética *se R é recursiva ou se existe uma relação recursiva S tal que R possa ser obtida de S por alguma sequência finita de complementações e/ou projecções.*

Teorema 5.1.16 (Teorema da Projecção)**.** *Se R é recursivamente enumerável, então qualquer projecção de R é uma relação recursivamente enumerável. Em particular, se R é relação k-ária recursivamente enumerável, então $\{(x_1, \ldots, x_{k-1}) | (\exists x_k) R(x_1, \ldots, x_k)\}$ é recursivamente enumerável.*

Segue-se uma importante caracterização desta hierarquia:

Teorema 5.1.17. [HR87, Secção 14.1 Teorema I] *Uma relação n-ária R está na hierarquia aritmética sse R é relação recursiva, ou existe um m e uma relação $(n+m)$-ária recursiva S tal que*

$$R = \{(x_1, \ldots, x_n) | (Q_1 y_1) \cdots (Q_n y_n) S(x_1, \ldots, x_n, y_1, \ldots, y_n)\}$$

onde $Q_i \in \{\forall, \exists\}$, $1 \leq i \leq n$.

As expressões na forma da relação R do Teorema anterior chamam-se "*predicate form*". Chama-se prefixo de uma relação R definida por uma "*predicate form*"à sua sequência de quantificadores. Por exemplo, a relação $(\forall x_1).(\exists x_2)(\exists x_3)\ R(x_1, x_2, x_3, x_4)$, tem prefixo $\forall\exists\exists$. Define-se *número de alternância do prefixo* da relação R como sendo o número de pares de quantificadores adjacentes distintos do prefixo de R.

Definição 5.1.18. *Para $n > 0$, um Σ^0_n-prefixo é um prefixo que começa com um quantificador $\exists$ e tem número de alternância $n-1$.*

Para $n > 0$, um Π^0_n-prefixo é um prefixo que começa com um quantificador $\forall$ e tem número de alternância $n-1$.

Um prefixo Π^0_0 é um prefixo vazio (sem quantificadores).

Assim, um *predicate form* tem uma Π^0_n-forma se tem um Π^0_n-prefixo, e analogamente se define Σ^0_n-forma. O supra-índice nesta notação refere-se à hierarquia de conjuntos em questão, onde, por exemplo, o índice 0 se refere à *hierarquia aritmética* e o 1 à *analítica*. Uma vez que no presente texto se trabalha exclusivamente a hierarquia aritmética, o supra-índice destas classes será sempre o 0.

Definição 5.1.19. *A classe Σ^0_n é a classe de todas as relações definíveis por uma Σ^0_n-forma, e a classe Π^0_n é a classe de todas as relações definíveis por uma Π^0_n-forma.*

Tem-se da definição anterior que uma relação R está na hierarquia aritmética se e só se existe um n tal que R seja membro de Σ^0_n ou de Π^0_n.

O Teorema que se segue, estabelece algumas propriedades importantes das classes da hierarquia aritmética:

Teorema 5.1.20 (Teorema de Kleene)**.** *Para $n \geq 1$:*

1. $\Delta^0_n \subset \Sigma^0_n$ *e* $\Delta^0_n \subset \Pi^0_n$;
2. $\Sigma^0_n \subset \Sigma_{n+1}$ *e* $\Pi^0_n \subset \Pi^0_{n+1}$;
3. $\Sigma^0_n \cup \Pi^0_n \subset \Delta_{n+1}$.

A demonstração do resultado pode ser encontrada em [DSW94, Teorema 5.10].

Teorema 5.1.21. *Para qualquer $n > 0$, $(\Sigma^0_n - \Pi^0_n) \neq \emptyset$.*

Tem-se, assim, que todas as relações recursivas estão em $\Sigma_0 = \Pi_0$ e pelo Teorema da Projecção, que todas as relações recursivamente enumeráveis estão em Σ_1. Pelos Teoremas anteriores, tem-se que a classe das relações co-recursivamente enumeráveis está em Π^0_1, e assim, a classe Π^0_2 estende as classes das relações recursivamente enumeráveis, e co-recursivamente enumeráveis. Do Teorema anterior sai que:

Facto 1. *Existem relações em Π^0_2 que não são recursivamente enumeráveis nem co-recursivamente enumeráveis.*

E portanto:

Facto 2. *Existem conjuntos em* Π_2^0 *que não são recursivamente enumeráveis nem co-recursivamente enumeráveis.*

Definição 5.1.22. [Sim08] *Dado um* $n \geq 1$*, um conjunto* $B \subseteq \mathbb{N}$ *diz-se um conjunto* Π_n^0-hard *se para todo o conjunto* $A \in \Pi_n^0$*,* $A \leq B$*. Adicionalmente, se* $B \in \Pi_n^0$ *diz-se que* B *é um conjunto* Π_n^0*-completo.*

Observe-se que é possível reduzir um problema Π_n^0-completo a um problema Π_n^0-hard, isto é, um problema com a mesma dificuldade mas, possivelmente, noutra classe da hierarquia aritmética.

5.1.4 Máquina Turing e Funções $\mathcal{T}$-Computáveis

Tal como foi referido na Secção 5.1, conhecem-se várias caracterizações relativas à computabilidade de funções, sendo provavelmente a que se apresenta de seguida, a que ilustra de forma mais intuitiva, o carácter "mecânico"do procedimento algorítmico. Uma máquina de Turing $\mathcal{T}$ pode ser vista como um engenho mecânico de leitura/escrita (*cabeça da máquina*) associado a uma *fita* de tamanho infinito (em ambas as direcções), particionada em *células* de igual tamanho, cada qual com um símbolo do alfabeto $\{0, 1\}$ inscrito (onde o símbolo 0 representa uma célula em branco). A cabeça da máquina pode aceder a uma célula de cada vez, sendo esta denominada a cada instante da execução por *célula actual* de $\mathcal{T}$. Uma máquina $\mathcal{T}$ é definida por um conjunto finito de *estados internos* $Q = \{q_0, \ldots, q_n\}$ e por uma *unidade de processamento*, que consiste num conjunto finito de regras deterministas denominadas por *instruções*, que decidem cada acção a executar pela máquina, em função do seu estado interno e célula actuais. Uma acção determina uma possível alteração do estado interno de $\mathcal{T}$ e a execução de um dos seguinte comandos:

1. escrever "1" na célula;
2. escrever "0" na célula;
3. andar uma célula para a direita " $\rightarrow$ ";
4. andar uma célula para a esquerda " $\leftarrow$ ".

Uma instrução é descrita por um quadruplo $\langle q, c, o, q' \rangle$ (normalmente representadas na forma $qcoq'$ que significa que "*se* $\mathcal{T}$ *está no estado interno* q *e com célula actual de valor* c*, então executa* o *e passa para o estado interno* q'", onde o representa um dos comandos em cima descritos. Por exemplo, a instrução $q_1 1 0 q_2$, significa que se a máquina estiver no estado q_1, com célula actual de valor 1 então, escreve (na célula actual) 0, passa ao estado q_2 e permanece na mesma posição. A cada máquina estão associados também um *estado inicial* q_s e um *estado terminal* q_h, que representam os estados a partir dos quais a máquina deve iniciar e terminar a sua execução respectivamente. Uma vez atingido o estado terminal q_h, a máquina pára a sua execução (contrariamente ao caso dos autómatos (Exemplo 2.1.6), nos quais, encontrado um estado terminal, é possível transitar para um outro que não o seja). Formalmente:

Definição 5.1.23 (Máquina de Turing)**.** *Dado um conjunto finito de* estados internos Q *com dois estados* q_s *e* q_h *denominados por* estados inicial *e* terminal *respectivamente, sejam* $B = \{0, 1\}$ *um conjunto de símbolos denominado por* alfabeto *e um conjunto* $C = \{0, 1, \rightarrow, \leftarrow\}$ *denominado por* conjunto de comandos*. Uma* máquina de Turing *é uma função de* $Q \times B$ *para* $Q \times C$*.*

Várias generalizações deste conceito estão presentes na literatura, nomeadamente o caso onde se consideram diferentes alfabetos de escrita B (cf. [Sip96, Definição 3.1]), onde se consideram máquinas com várias fitas (cf. [Sip96, Secção 3.2]), onde se consideram máquinas com várias cabeças (cf. [HUM01, Secção 7.5]), etc. A formalização aqui apresentada pode ser encontrada em [HR87, Secção 1.5].

Qualquer passo da execução de uma máquina de Turing é descrito pelo seu estado interno q_i, pela célula actual (posição da cabeça da máquina) e pela string inscrita na fita da máquina nesse instante. Esta informação denomina-se por *descrição instantânea ou configuração da máquina*. Uma descrição instantânea de uma máquina pode-se representar na forma $\cdots 0 a_1 \cdots a_n q b_1 b_2 \cdots b_n 0 \cdots$ que significa que a célula actual é a que está à direita de q, para q estado actual da máquina com string inscrita $\cdots 0 a_1 \cdots a_n b_1 b_2 \cdots b_n 0 \cdots$. A execução de uma máquina fica completamente descrita pela sucessão das suas configurações.

Tal como acontece com os *autómatos* (ver Exemplo 2.1.6), estas máquinas podem ser utilizadas no reconhecimento de linguagens, sendo que a sua principal função é a de "computar"funções parcialmente recursivas (Secção 5.1.4). Esta computação faz-se da seguinte forma: dado um input $(x_1, \ldots, x_n)$, escreve-se na fita de $\mathcal{T}$ (inicialmente

preenchida exclusivamente de 0's), uma string de $x_1 + 1$ símbolos 1 seguidos de um símbolo 0, seguido de $x_2 + 1$ símbolos 1 com outro 0 e assim sucessivamente (no seguimento do texto, quando se diz "uma máquina de Turing com input $(x_1, \ldots, x_n)$", pretende-se dizer "uma máquina de Turing com um input"construído desta forma). A execução de $\mathcal{T}$, iniciada na primeira célula (à esquerda) diferente de 0 e a partir do estado $q_s \in Q$, gera uma sucessão de acções, que caso atinja o estado q_h, inscreve na fita, o resultado de $f(x_1, \ldots, x_n)$ (sob a forma da quantidade de 1's inscritos).

Por outro lado, dada uma função parcialmente recursiva $f : \mathbb{N}^n \to \mathbb{N}$, existe (pelo menos) uma máquina de Turing $\mathcal{T}_f$ que a compute, isto é, que dado um input $(x_1, \ldots, x_n)$ retorne o resultado de $f(x_1, \ldots, x_n)$ no caso de f estar definida em $(x_1, \ldots, x_n)$, ou no caso contrário, continue para sempre a sua execução. Observe-se ainda que uma mesma máquina de Turing $\mathcal{T}$ pode calcular diversas funções de diferentes aridades, tal como se pode ver no Exemplo que se segue:

Exemplo 5.1.24. *Considere-se a máquina de* Turing $\mathcal{T}$ *definida pela seguinte unidade de processamento:*

$$\langle q_s, 1, \rightarrow, q_s\rangle$$
$$\langle q_s, 0, \rightarrow, q_1\rangle$$
$$\langle q_1, 1, \leftarrow, q_4\rangle$$
$$\langle q_1, 0, \leftarrow, q_2\rangle$$
$$\langle q_2, 0, \leftarrow, q_3\rangle$$
$$\langle q_3, 1, 0, q_h\rangle$$
$$\langle q_4, 0, 1, q_5\rangle$$
$$\langle q_5, 1, \rightarrow, q_5\rangle$$
$$\langle q_5, 0, \leftarrow, q_6\rangle$$
$$\langle q_6, 1, 0, q_7\rangle$$
$$\langle q_7, 0, \leftarrow, q_8\rangle$$
$$\langle q_8, 1, 0, q_h\rangle$$

Esta máquina computa as funções $f(x, y) = x + y$, *e* $f(x) = x$.

◇

Máquina Universal

Uma máquina de Turing $\mathcal{T}$ pode ser exclusivamente definida à custa da sua unidade de processamento, isto é, à custa de um conjunto de instruções expressas sob a forma de

quadruplos $\langle q, c, o, q' \rangle$, na medida em que este conjunto também define o conjunto Q dos estados internos de $\mathcal{T}$. Existem processos efectivos a partir dos quais se consegue codificar um qualquer conjunto deste tipo num natural n e vice versa, ou seja, é possível codificar qualquer máquina de Turing $\mathcal{T}$ num natural n assim como construir uma e uma só máquina $\mathcal{T}$ a partir de um natural n. Este processo é conhecido por Gödelização das máquinas de Turing. A escolha de um processo concreto para a Gödelização de máquinas não é aqui fixado, sendo a única condição necessária para o trabalho que aqui se apresenta, a existência de uma bijecção via codificação, entre os naturais e as partes do conjunto das instruções das máquinas de Turing. Os algoritmos de Gödelização são normalmente baseados na factorização de naturais em números primos. Em [Bil03, Capítulo 14] e [HUM01, Secção 8.3], descrevem-se dois processos distintos para a Gödelização de máquinas de Turing. Processos similares a este são elaborados para outros modelos de computação. Por exemplo, em [Ser93], descreve-se este processo para as máquinas URM, em [DSW94] é descrito para uma linguagem $\mathfrak{S}$, e em [End01, HR87] é descrito para as funções parcialmente recursivas. Esta técnica está na base de importantes resultados ao nível do estudo da computabilidade de funções e equivalência entre modelos de computação.

Dado que se pode construir qualquer máquina (de forma algorítmica) a partir de um natural n, é possível construir uma máquina que a partir desse natural e de um determinado input (para a máquina de número n), simule a execução da *máquina de Turing* de número de Gödel n a partir desse input. Este é o princípio do conceito de *máquina* ou *programa universal*:

Definição 5.1.25. *Sejam $m, n \in \mathbb{N}_0$. Define-se a função $\Phi^{(n+1)} : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$, como a função de aridade n calculada pela máquina de Turing cujo número de Gödel é m. A função $\Phi^{(n+1)}(x_1, \ldots, x_n, m)$ também se representa por $\Phi_m^{(n)}(x_1, \ldots, x_n)$.*

A máquina de Turing $\mathcal{T}$ que computa a função $\Phi^{(n+1)}$ é também conhecida por *Máquina de Turing Universal*. É possível portanto, a partir da sequência

$$\Phi^{(n)}(x_1, \ldots, x_n, 0), \Phi^{(n)}(x_1, \ldots, x_n, 1), \ldots$$

enumerar todas as funções parcialmente recursivas de n variáveis. Define-se de seguida um predicado que é frequentemente utilizado para fazer demonstrações nesta área da matemática.

Definição 5.1.26 ([DSW94])**.** *Sejam $x_i, y, t \in \mathbb{N}$ para $i = 1, \ldots, n$. Define-se o predicado $STP^{(n)}$ da seguinte forma:*

$$STP^{(n)}(x_1, \ldots, x_n, y, t)$$

sse

a máquina de Turing de número de Gödel y pára depois de t ou menos passos, a partir do input $(x_1, \ldots, x_n)$.

Proposição 5.1.27. [DSW94, Capítulo 4, Teorema 3.2] *Para todo $n > 0$, o predicado $STP^{(n)}(x_1, \ldots, x_n, y, t)$ é um predicado (primitivo) recursivo.*

Outro resultado que aqui se utiliza é o seguinte:

Teorema 5.1.28. *Para quaisquer $m, n > 0$, existe uma função total $\mathcal{T}$-computável $S_m^n(u_1, \ldots, u_n, y) \in [\mathbb{N}_0^{(n+1)} \to \mathbb{N}_0]$ tal que*

$$\Phi^{(m+n)}(x_1, \ldots, x_m, u_1, \ldots, u_n, y) = \Phi^{(m)}(x_1, \ldots, x_m, S_m^n(u_1, \ldots, u_n, y)).$$

Este Teorema aparece na literatura como o *resultado s-m-n*, e encontra-se demonstrado em [HR87] Secção 1.5, Teorema V.

Corolário 5.1.29. [DSW94, Capítulo 8, Teorema 2.5] *Segundo as hipóteses do Teorema anterior, caso Φ_y seja uma função total, então $S_m^n(u_1, \ldots, u_n, y) = S_m^n(u'_1, \ldots, u'_n, y)$ implica que $u_1 = u'_1, \ldots, u_n = u'_n$.*

5.2 A Satisfação Comportamental é Π_2^0-Hard

5.2.1 Problema da *Totalidade*

Apresenta-se nesta Secção um problema que, pelas suas propriedades, desempenha um papel central na demonstração da incompletude das lógicas comportamentais: o problema da *Totalidade*. Pela teoria da recursividade, um dado "problema"pode ser visto como o problema de reconhecer, de forma efectiva, os elementos de um conjunto ou relação, isto é, traduz-se no estudo da natureza da função característica do conjunto das soluções do "problema". Neste contexto, tem-se que um "problema"é solúvel se o conjunto das suas soluções for recursivo. Assim, classificar um problema na hierarquia aritmética, significa classificar o seu conjunto de soluções nessa mesma hierarquia. Dada uma máquina de Turing $\mathcal{T}$, define-se o problema da Totalidade de $\mathcal{T}$ da seguinte forma:

Definição 5.2.1. *Seja $\mathcal{T}$ uma máquina de Turing. O problema $TOTALIDADE_{\mathcal{T}}$ consiste no seguinte:*

- *INPUT: Um inteiro $k \geq 0$*
- *OUTPUT: A máquina de Turing $\mathcal{T}$ pára em todos os inputs 1^j01^k, para $j \geq 0$?*

Quando para um dado k se tem que a máquina de Turing $\mathcal{T}$ pára em todos os inputs 1^j01^k, para todo o $j \geq 0$, escreve-se que $TOTALIDADE_{\mathcal{T}}(k)$ é positivo. Segue uma importante propriedade:

Teorema 5.2.2. *Existem máquinas de Turing $\mathcal{T}$ nas quais $TOTALIDADE_{\mathcal{T}}$ é Π_2^0-completo.*

Demonstração. Sejam $\mathcal{T}$ uma máquina de Turing, $\Psi(j,k)$ a função (parcialmente recursiva) computada por $\mathcal{T}$ a partir do input 1^j01^k e Φ o programa universal. Para a função $\Psi(j,k)$, existe um número de Gödel y tal que

$$\Psi(j,k) = \Phi((j,k),y) = \Phi_y(j,k),$$

e assim, pode-se traduzir o problema da Totalidade pelo predicado

$$p(k) := (\forall j)(\exists t)STP((j,k),y,t).$$

Uma vez que $STP((j,k),y,t)$ é um predicado recursivo (Proposição 5.1.27), segue da Definição 5.1.19 que $p(k) \in \Pi_0^2$, e portanto $TOTALIDADE_{\mathcal{T}} \in \Pi_0^2$, qualquer que seja a máquina de Turing $\mathcal{T}$.

Considere-se agora o caso em que $\Psi = \Phi$, isto é, o caso de $\mathcal{T}$ ser a máquina universal. Tem-se por definição de programa universal que $\Phi(j,k) = \Phi_k(j)$ é a função parcialmente recursiva de número k, para k número de Gödel para programas universais. Assim, o problema da Totalidade é positivo quando $\Phi_k(j)$ termina para qualquer $j \geq 0$, isto é, quando $\Phi_k(j)$ é função total. Desta forma, o problema da $TOTALIDADE_{\Phi}(k)$ é positivo para todos os k's do conjunto $A = \{k \in \mathbb{N} | \Phi_k(j) \text{ é total}\}$.

Pretende-se agora demonstrar que o conjunto A é Π_2^0-completo. Tem-se, que

$$A = \{k \in \mathbb{N} | \Phi_k(j) \text{ é total}\}$$

$$= \{k \in \mathbb{N} | (\forall j)\Phi_k(j) \downarrow\}$$

$$= \{k \in \mathbb{N} | (\forall j)(\exists y)STP(j,y,t)\}.$$

Pelo Teorema 5.1.27, tem-se que o predicado STP é predicado primitivo recursivo e, portanto, tem-se que $P(k) := (\forall j)(\exists t)STP(j,k,t) \in \Pi_0^2$. Logo $A \in \Pi_2^0$.

Considere-se agora o conjunto $B = \{w \in \mathbb{N} | (\forall x)(\exists y)R(x,y,w)\}$, para $R(x,y,w)$ predicado recursivo. Pela Definição 5.1.8, tem-se que a função $h(x,w) = \min_y R(x,y,w)$ é uma função parcialmente recursiva. Seja e o número de Gödel para o programa que computa a função h. Então,

$$(\exists y)R(x,y,w) \Leftrightarrow h(x,w) \downarrow$$

$$\Leftrightarrow \Phi(x,w,e) \downarrow,$$

pelo Teorema 5.1.28,

$$\Phi(x, S_1^1(w,e)) \downarrow .$$

Assim sendo, tem-se que:

$$w \in B \Leftrightarrow (\forall x)(\exists y)R(x,y,w)$$

$$\Leftrightarrow (\forall x)[\Phi(x, S_1^1(w,e)) \downarrow]$$

$$\Leftrightarrow \Phi(x, S_1^1(w,e)) \text{ é total}$$

$$\Leftrightarrow S_1^1(w,e) \in A$$

tem-se pelo Corolário 5.1.29 que S_1^1 é injectiva, pelo que, sai directamente da definição de $\leq_1$ que $B \leq_1 A$, e portanto, A é Π_2^0-completo. Fica assim provada a existência de máquinas de Turing $\mathcal{T}$ (pelo menos a máquina universal), onde o problema $TOTALIDADE_{\mathcal{T}}$ é Π_2^0-completo. □

5.2.2 A Satisfação Observacional é Π_2^0-Hard

Considere-se uma máquina de Turing $\mathfrak{T}$ tal que o problema $TOTALIDADE_{\mathfrak{T}}$ seja Π_2^0-completo (Teorema 5.2.2). Em função desta máquina, define-se de seguida uma especificação equacional, por forma a simular algebricamente a sua execução. Consideram-se, para tal, uma operação `blank` para preencher a fita da máquina de 0's, as operações $\mathtt{0}_r$ e $\mathtt{1}_r$ para inscrever os símbolos 0 e 1 à direita de uma *string*, as operações $\mathtt{0}_l$ e $\mathtt{1}_l$ para os escrever à esquerda, para cada estado q uma operação `q` para executar os passos da máquina em função da sua unidade de processamento e as operações de resultado booleano `willstop` e `always` que, dada uma descrição instantânea da máquina, indicam se a sua execução pára ou decorre indeterminadamente. Assim:

Spec INCOMPLETA =

[GEN]

h;

v;

[OBS]

v;

[OP]

true: -> v;

false: -> v;

willstop: h -> v;

blank: h -> h;

always: h -> h;

more: h -> h;

q: h -> h, para cada $q \in Q$ (uma função para cada estado da máquina);

0_l : h -> h;

0_r : h -> h;

1_l : h -> h;

1_r : h -> h;

[AX]

1. ($\forall$x:h). blank(m(x))=blank(x) para todo o método m;
2. ($\forall$x:h). m(always(x))=always(x) para todo o método m $\neq$ blank;
3. ($\forall$x:h). m(q(x))=always(x) para todo o método m $\neq$ blank e q tal que m $\neq$ more ou q $\neq$ q_s;
4. ($\forall$x:h).more(q_s(x))=q_s(1_r(x));
5. ($\forall$x:h).b'$_l$(b$_r$(x))=b$_r$(b'$_l$(x)) para b $\in \{1, 0\}$;
6. ($\forall$x:h).0_d(blank(x))=blank(x) para d $\in \{l, r\}$;
7. ($\forall$x:h).willstop(always(x))=true;

8. $(\forall$`x:h`$)$.`willstop(`$\mathtt{q}_h$`(x))=true`;

9. Para toda a instrução $qbcq'$ da unidade de processamento de $\mathfrak{T}$ adicionar:

   (a) $(\forall$`x:h`$)$.`willstop(q(`$\mathtt{b}_r$`(x)))=willstop(q'(`$\mathtt{c}_r$`(x)))` caso $c = 0$ ou $c = 1$;

   (b) $(\forall$`x:h`$)$.`willstop(q(`$\mathtt{b}_r$`(x)))=willstop(q'(`$\mathtt{b}_l$`(x)))` caso $c = \rightarrow$;

   (c) $(\forall$`x:h`$)$.`willstop(q(b'`$_l$`(`$\mathtt{b}_r$`(x)))=willstop(q'(b'`$_r$`(`$\mathtt{b}_r$`(x)))` para $b' = 0$ e $b' = 1$, caso $c = \leftarrow$.

Por exemplo, a representação da configuração $\cdots 0b_1 \cdots b_n 0 \cdots$ para $b_i \in \{0, 1\}$, faz-se a partir do termo $\mathtt{b}_{1_r}(\cdots(\mathtt{b}_{n_r}(\mathtt{blank(x)}))\cdots)$, da seguinte forma: limpa-se a fita `x` (preenche-se de 0's) pela função `blank`, e introduz-se o input, símbolo a símbolo, pelas operações $\{\mathtt{1}_r, \mathtt{0}_r\}$.

Observe-se que a especificação anterior apresenta uma estrutura muito simples, uma vez que só apresenta um género observável, um género não observável e operações unárias. Este factor é importante, na medida em que esta especificação é capturada por diversos formalismos da semântica observacional de sistemas presentes na literatura. Tem-se, por exemplo, que tanto está de acordo com os formalismos aqui apresentados (Capítulo 3) como também por outros mais restritivos, como é o caso das *hidden algebras* apresentadas em [GD94b], onde se impõe que as operações da assinatura possuam no máximo um argumento de género não observável.

Lema 5.2.3. *Se $\mathfrak{T}$ pára a partir do input $\mathtt{b}^1 \cdots \mathtt{b}^n$, então*

$$\mathtt{INCOMPLETA} \models (\forall \mathtt{x:h}).\mathtt{willstop}(\mathtt{q}_s(\mathtt{b}^1_r(\cdots(\mathtt{b}^n_r(\mathtt{blank(x)})))) = \mathtt{true}.$$

Demonstração. Se a máquina $\mathfrak{T}$ pára a partir do input $b^1 \cdots b^n$, então há uma sucessão de descrições instantâneas de $\mathfrak{T}$ começados por $\cdots 0 q_s b^1 b^2 \cdots b^n 0 \cdots$ e terminadas por $\cdots b' q_h b'' \cdots$. Note-se que se se fizer, em cada passo, um rearranjo adequado dos métodos à esquerda dos $\mathtt{q}_i$, $i \in \{1, \ldots, n\}$, (pelo axioma 5.), os axiomas do ponto 9. simulam correctamente todos os passos efectuados pela máquina. Pela aplicação sucessiva da transitividade e destes axiomas chega-se a determinada altura da execução de $\mathfrak{T}$ a:

$$\mathtt{INCOMPLETA} \models (\forall \mathtt{x:h}).\mathtt{willstop}(\mathtt{q}_s(\mathtt{b}^1_r(\cdots(\mathtt{b}^n_r(\mathtt{blank(x)})))) = \mathtt{willstop}(\mathtt{q}_h(\mathtt{t}(\mathtt{blank(x)})))),$$

para algum $\mathtt{t}$ composto por métodos de $\{\mathtt{0}_l, \mathtt{0}_r, \mathtt{1}_l, \mathtt{1}_r\}$, e assim, por (8) tem-se que

$$\mathtt{INCOMPLETA} \models (\forall \mathtt{x:h}).\mathtt{willstop(q}_s\mathtt{(b}_{1_r}\mathtt{(}\cdots\mathtt{(b}_{n_r}\mathtt{(blank(x)))))=true}$$

tal como se pretendia. □

Considere-se agora a seguinte $Sig(\mathtt{INCOMPLETA})$-álgebra A:

- $A_v = \{\mathtt{true, false}\}$;
- $A_h = ((Q \cup \bot)) \times String \times String) \cup \Box$, onde $\Box$ é um novo elemento especial;
- $\mathtt{willstop}^A(X) = \mathtt{true}$, caso $X = \Box$ ou $X = (\bot, \mathtt{S,S'})$ ou se a máquina de Turing $\mathfrak{T}$ pára a partir do input $(\mathtt{q}, \cdots 0 \cdots 0\mathtt{S}|\mathtt{S'}0 \cdots 0 \cdots)$; é $\mathtt{false}$ noutros casos;
- $$\mathtt{more}^A(X) = \begin{cases} \Box \text{ caso } X = \Box \text{ ou } X = (q, S, S') \text{ para } q \neq q_s \\ (q_s, S, 1S') \text{ caso } X = (q_s, S, S') \end{cases}$$
- $\mathtt{blank}^A(X) = (\bot, \epsilon, \epsilon)$, para todo o $X \in A_h$;
- $$\mathtt{q}^A(X) = \begin{cases} \Box \text{ caso } X = \Box \text{ ou } X = (q', S, S') \text{ para } q, q' \in Q \\ (q, S, S') \text{ caso } X = (\bot, S, S') \\ \Box \text{ noutros casos} \end{cases}$$
- $\mathtt{b}^A(X) = \Box$ para todo $b \in \{0_l, 0_r, 1_l, 1_r\}$ caso $X = \Box$ ou $X = (q, S, S')$ para todo o $q \in Q$;
- $\mathtt{b}_l^A((\bot, S, S')) = (\bot, Sb, S')$ para todo $b \in \{0, 1\}$ e para quaisquer *strings* S, S' tais que $b \neq 0$ ou $S \neq \epsilon$;
- $\mathtt{b}_r^A((\bot, S, S')) = (\bot, S, bS')$ para todo $b \in \{0, 1\}$ e S, S'Strings tais que $b \neq 0$ ou $S' \neq \epsilon$;
- $\mathtt{0}_l^A((\bot, \epsilon, S')) = (\bot, \epsilon, S')$ para qualquer string S';
- $\mathtt{0}_r^A((\bot, S, \epsilon)) = (\bot, S, \epsilon)$ para qualquer string S'.

Facto 3. *A especificação* $\mathtt{INCOMPLETA}$ *é consistente na medida em que* $A \in [\![\mathtt{INCOMPLETA}]\!]$.

Teorema 5.2.4. *Dado o inteiro $k \geq 0$, seja e_k a equação*

$$(\forall \mathtt{x{:}h}).\mathtt{q}_s(\mathtt{0}_r(\mathtt{1}_r(\mathtt{1}_r(\cdots(\mathtt{1}_r(\mathtt{blank(x)}))\cdots)))) = \mathtt{always(x)},$$

onde $\mathtt{1}_r$ ocorre k vezes. São equivalentes as seguintes afirmações

1. *$TOTALIDADE_{\mathfrak{T}}(k)$ é positivo;*
2. $\mathtt{INCOMPLETA} \models_{\approx_{Obs}} e_k$.

Demonstração. (1. $\Rightarrow$ 2.) Dada a natureza das operações de $Sig(\mathtt{INCOMPLETA})$, tem-se que esta só admite contextos observáveis da forma $\mathtt{willstop}(\mathtt{m}_1(\cdots(\mathtt{m}_n(\mathtt{z}_h)))$, para todo $\mathtt{m}_i$ método de INCOMPLETA. Com vista ao absurdo, considere-se que existe um contexto $c \in \mathcal{C}^{Obs}_{Sig(\mathtt{INCOMPLETA})}$ tal que não se verifique

$$\mathtt{INCOMPLETA} \models (\forall \mathtt{x{:}h}).\mathtt{c}[\mathtt{q}_s(\mathtt{0}_r(\mathtt{1}_r(\mathtt{1}_r(\cdots(\mathtt{1}_r(\mathtt{blank(x)}))\cdots))))] = \mathtt{c[always(x)]}.$$

Então:

- caso exista um $i \in \{1, \ldots, n\}$ tal que $\mathtt{m}_i = \mathtt{blank}$, pela aplicação de (1) tem-se $\mathtt{willstop}(\mathtt{m}_1(\cdots(\mathtt{blank}(\cdots(\mathtt{m}_n(\mathtt{q}_s(\mathtt{0}_r(\mathtt{1}_r(\mathtt{1}_r(\cdots(\mathtt{1}_r(\mathtt{blank(x)}))\cdots)))))))) = \mathtt{willstop}(\mathtt{m}_1(\cdots(\mathtt{blank(x)}))))$, e pela mesma razão

  $$\mathtt{c(always(x))} = \mathtt{willstop}(\mathtt{m}_1(\cdots(\mathtt{blank(x)})))$$

  o que é contraditório! Logo, não existe nenhum $i \in \{1, \ldots, n\}$ tal que $\mathtt{m}_i = \mathtt{blank}$;

- caso exista um $i \in \{1, \cdots, n\}$ tal que $\mathtt{m}_i = \mathtt{always}$, então, qualquer que seja o termo t do género h,

  $$\mathtt{willstop}(\mathtt{m}_1(\cdots(\mathtt{always}(\cdots(\mathtt{m}_n(\mathtt{t}))\cdots))\cdots)) = \mathtt{willstop}(\mathtt{m}_1(\cdots(\mathtt{always(x)}\cdots))),$$

  e mais uma vez

  $$(\forall \mathtt{x{:}h}).\mathtt{c}[\mathtt{q}_s(\mathtt{0}_r(\mathtt{1}_r(\mathtt{1}_r(\cdots(\mathtt{1}_r(\mathtt{blank(x)}))\cdots))))] = \mathtt{c[always(x)]}.$$

  Chega-se assim a novamente a uma contradição, e portanto, não existe nenhum $i \in \{1, \ldots, n\}$ tal que $\mathtt{m}_i = \mathtt{always}$;

- tem-se pelos axiomas 2. e por 7. que, qualquer que seja o contexto c,

$$c[\texttt{willstop(x)}] = true. \tag{5.3}$$

  Por outro lado, tem-se por (3) que caso $\texttt{m}_n \neq \texttt{more}$, $\texttt{willstop}(\texttt{m}_1(\cdots(\texttt{m}_n(\texttt{q}_s(\cdots))$ $= \texttt{willstop}(\texttt{m}_1(\cdots(\texttt{m}_{n-1}(\texttt{always}(\cdots))$, e por (2) e (7) resulta

$$\texttt{willstop((always(x))=true}.$$

  Chega-se assim a novamente a uma contradição, e portanto $\texttt{m}_n$=more.

- caso exista um $i < n$ tal que $\texttt{m}_i \neq \texttt{more}$, tem-se

$$\texttt{willstop}(\texttt{m}_1(\cdots(\texttt{m}_i(\texttt{more}(\cdots(\texttt{more}(\texttt{q}_s(0_r(1_r(\cdots))))\cdots))\cdots))$$

$$= \texttt{willstop}(\texttt{m}_1(\cdots(\texttt{m}_i(\texttt{q}_s(1_r(\cdots(1_r(0_r(1_r(\cdots))))\cdots))\cdots))$$

  para $n-i$ ocorrências de 1_r antes do método 0_r, e por (5.3), chega-se novamente a uma contradição.

Fica assim provado que o contexto c tem que ser da forma $\texttt{willstop}(\texttt{more}(\cdots(\texttt{more}(\texttt{z}_h)))$, para n ocorrências do método more, e pela aplicação sucessiva do axioma 4. obtém-se

$$\texttt{willstop}(\texttt{more}(\cdots(\texttt{more}(\texttt{q}_s(0_r(1_r(\cdots(\texttt{blank(x)})))))$$

$$= \texttt{willstop}((\texttt{q}_s(1_r(\cdots(1_r(0_r(1_r(\cdots(\texttt{blank(x)})))))).$$

Tem-se por hipótese que $TOTALIDADE_{\mathfrak{T}}(k)$ é positivo, e portanto, sai do Teorema 5.2.1 que

$$\texttt{willstop}((\texttt{q}_s(1_r(\cdots(1_r(0_r(1_r(\cdots(\texttt{blank(x)})))))\texttt{=true}.$$

Por (5.3), chega-se a uma nova contradição!

Portanto não existe nenhum contexto c tal que não se verifique

$$\texttt{INCOMPLETA} \models (\forall \texttt{x:h}).\texttt{c}[\texttt{q}_s(0_r(1_r(1_r(\cdots(1_r(\texttt{blank(x)}))\cdots)))))]\texttt{=c[always(x)]},$$

isto é, $\texttt{INCOMPLETA} \models_{\approx_{Obs}} e_k$ tal como se pretendia.

(1. $\Leftarrow$ 2.) Tem-se que $\texttt{INCOMPLETA} \models_{\approx_{Obs}} e_k$, e portanto, para o caso particular do contexto $\texttt{willstop}(\texttt{z}_h)$, tem-se que

$$\texttt{INCOMPLETA} \models (\forall \texttt{x:h}).\texttt{willstop}(\texttt{q}_s(0_r 1_r(\cdots(1_r(\texttt{blank(x)})))))=$$

$$\texttt{willstop(always(x))},$$

e pelo axioma 7.,

$$\texttt{INCOMPLETA} \models (\forall \texttt{x:h}).\texttt{willstop}(\texttt{q}_s(\texttt{0}_r\texttt{1}_r(\cdots(\texttt{1}_r(\texttt{blank(x)})))) = \texttt{true}.$$

Tem-se também do Facto 3 que A é modelo de INCOMPLETA, e portanto

$$\texttt{willstop}^A((\texttt{q}_s, \epsilon, 1^j01^k)) = \texttt{true}.$$

Pela definição de A, a máquina de Turing $\mathfrak{T}$ pára para o input 1^j01^k, isto é, o problema $TOTALIDADE_{\mathfrak{T}}(k)$ é positivo, tal como se pretendia demonstrar. □

Chega-se desta forma ao resultado central da Secção:

Facto 4. *Em todos os formalismos da semântica observacional de sistemas abordados no texto existem especificações onde o problema da satisfação observacional é Π_2^0-Hard. Assim sendo, não existem algoritmos que nelas verifiquem todas as proposições observacionalmente verdadeiras, nem que nelas refutem todas as que não o sejam.*

A demonstração do Teorema sai directamente do facto do problema da Totalidade se reduzir ao problema da satisfação observacional. Assim, uma vez que $TOTALIDADE_{\mathfrak{T}}$ é Π_2^0-completo, tem-se que o problema da satisfação é Π_2^0-hard. Dado que a satisfação observacional é um caso particular da satisfação comportamental, tem-se também que:

Corolário 5.2.5. *O problema da satisfação comportamental em algumas especificações é Π_2^0-Hard. Assim sendo, não existem algoritmos que nelas verifiquem todas as proposições comportamentalmente verdadeiras, nem que nelas refutem todas as que não o sejam.*

Capítulo 6

Conclusões e Desenvolvimentos Futuros

O objectivo proposto para a presente tese foi o de estudar, numa perspectiva algébrica, as práticas de verificação e desenvolvimento formal de software com dados encapsulados, via relação de *igualdade observacional*.

Procurou-se neste trabalho apresentar os assuntos em estudo, a um nível o mais geral possível, por forma a poderem ser utilizados em diferentes contextos de aplicação. Neste sentido, escolheu-se a abordagem de *Bidoit* e de *Hennicker* para a apresentação e formalização dos conceitos centrais da semântica observacional de sistemas (Capítulo 3). Nomeadamente, apresentou-se esta teoria para o caso das *Igualdades Comportamentais* em especificações estruturadas, contrariamente à maior parte das abordagens, que estudam exclusivamente o caso da *Igualdade Observacional* em especificações *flat*. Com o mesmo intuito, estudaram-se algumas adaptações e generalizações do conceito de processo de refinamento, nomeadamente a sua generalização aos σ-refinamentos e aos σ-refinamentos observacionais. Relativamente ao estudo deste último tópico, fez-se ainda a caracterização do caso onde se admitiam variações no conjunto dos géneros observáveis durante o processo de refinamento e demonstrou-se um resultado pelo qual, a partir de um refinamento observacional a respeito de uma determinada igualdade observacional, se constrói um outro refinamento sobre a mesma especificação, a respeito de uma relação de igualdade observacional mais pequena. O estudo deste tópico é feito exclusivamente para o caso das especificações e equacionais.

Outro objectivo central desta dissertação foi o de reunir e unificar diferentes resultados

relativos ao tema da tese, estudados por diferentes abordagens em contextos distintos. Neste sentido, deu-se especial atenção aos contributos das abordagens da Γ-*igualdade observacional* e da *Lógica Algébrica Abstracta*, referindo-se também outros importantes trabalhos sobre o tema, nomeadamente os desenvolvidos pelos grupos de *Diaconescu*, *Futsatsugui* e *Padawitz*. Também se apresentou, ainda que de forma sucinta, a outra grande abordagem à semântica observacional de sistemas presente na literatura - a abordagem *abstractor*. Esta abordagem, ao invés da abordagem central do texto, que se baseia na relação de *Igualdade Observacional* entre elementos, centra o seu estudo numa relação de equivalência entre álgebras. Apresentaram-se ainda alguns resultados para estabelecer a ponte entre esta estas duas abordagens.

Um outro tópico presente neste trabalho, ainda que não explicitamente mencionado, foi o paralelo entre o estudo da semântica observacional de sistemas e a *teoria dos autómatos*. Tal como se referiu na introdução da tese, objectos desenhados segundo o paradigma *OO* comportam-se como máquinas de transição de estados. De facto, a abordagem observacional não só transporta para a verificação standard (baseada na lógica equacional) propriedades e características destes sistemas, como também generaliza muitos dos conceitos estudados na teoria dos autómatos, nomeadamente o da *equivalência entre estados de um autómato*, o da *equivalência entre autómatos* (equivalência de *Nerode*) e de *autómato reduzido*. Todos estes casos apareceram no texto a título de exemplo de conceitos e resultados trabalhados (ver Exemplos 2.1.6, 2.1.12, 3.1.7, 3.1.13, 3.1.19 e 3.3.7).

6.1 Tópicos Para Trabalhos Futuros

6.1.1 Refinamentos por Tradução

Os conceitos de refinamento e de processo de refinamento passo-a-passo foram trabalhados neste texto em vários contextos e formalizações. Numa primeira fase, estudou-se o caso onde se impunha a preservação da assinatura entre passos de refinamento (Definição 2.4.1); numa segunda, as assinaturas das especificações podiam variar via morfismos (Definição 2.4.4); numa terceira, adequou-se o processo ao paradigma da observabilidade; e numa quarta caracterizou-se o caso onde eram admitidas variações no conjunto dos géneros observáveis durante o processo de refinamento (Secção 3.4.1). Percebeu-se também, aquando da apresentação destes conceitos que o crescente grau de

liberdade na noção de refinamento, induzido por estas diferentes formalizações, se mostra favorável às práticas de implementação de software por reutilização de componentes implementadas.

Sugere-se aqui uma alternativa à formalização do conceito de refinamento, baseada no conceito de *tradução lógica* usado no estudo da *Teoria Algébrica dos Sistemas Dedutivos* pela *Lógica Algébrica Abstracta* (cf. [BP89, BP98, BR03]). Os resultados e motivações que aqui se apresentam surgiram no desenvolvimento do presente trabalho de dissertação, tendo sido recentemente apresentadas com mais algum detalhe, em [Mad08]. Tal como na Secção 4.2, considere-se aqui o conjunto de fórmulas $Fm(\Sigma)$ como sendo o conjunto das equações e das equações condicionais sobre a assinatura Σ. Dadas duas assinaturas $\Sigma = (S, \Omega)$ e $\Sigma' = (S', \Omega')$ tais que $\Sigma \subseteq \Sigma'$, define-se $\Sigma - \Sigma'$-*tradução* τ como sendo uma S-família $(\tau_s(x : s, y : s))_{s \in S}$ tal que para cada $s \in S$, $\tau_s(x : s, y : s) = (\tau_{s,s'}(x : s, y : x))_{s' \in S'}$ é um S'-conjunto de Σ'-equações $\phi(x, y) = \psi(x, y)$ de género s' em duas variáveis do género s. Define-se τ-*tradução de uma* Σ-*equação* $t = t'$ *de género* s, em símbolos $\tau(t = t')$, como sendo o S'-conjunto de Σ'-equações $(\tau_{s,s'}(t, t'))_{s' \in S'}$ e estende-se esta definição a equações condicionais de forma natural. Esta definição aparece em [BP98] formulada para o caso das k-lógicas. Neste contexto, o conceito de tradução aparece como $(k - l)$-tradução, que significa uma tradução de uma lógica de dimensão k para uma outra de dimensão l da mesma assinatura. Um exemplo paradigmático de uma tradução deste tipo é a tradução do *cálculo proposicional clássico* na *teoria equacional das álgebras boolenas* [BP98, Exemplo 4.1.2].

Observe-se que um morfismo de assinaturas mapeia uma fórmula numa outra fórmula. No entanto, uma *tradução lógica* pode mapear uma fórmula num conjunto de fórmulas. Este facto, que vai ao encontro da desejada liberdade no conceito de refinamento, motivou a formalização do conceito de refinamento que aqui se sugere. Por forma a estabelecer uma relação entre as semânticas de uma especificação SP e da sua tradução diz-se que uma álgebra A' é um τ-modelo de SP se para toda a fórmula $\xi \in Fm(Sig(SP))$, $SP \models \xi$ implica $A' \models \tau(\xi)$ e, diz-se que uma tradução τ *interpreta* SP se existe uma especificação SP' tal que, para toda a fórmula $\xi \in Fm(\Sigma)$, $SP \models \xi$ sse $SP' \models \tau(\xi)$. Neste contexto mostra-se que se τ interpreta SP, então a especificação SP^τ é a τ-interpretação de SP com maior classe de modelos, onde SP^τ representa a especificação cuja classe de modelos é a classe dos τ-modelos de SP [Mad08, Teorema 3.6]. Mais, prova-se também que se SP for axiomatizada por um conjunto de equações e

de equações condicionais Φ então SP^τ é axiomatizada por $\tau(\Phi)$[Mad08, Teorema 3.7]. Tendo em conta esta "preservação de propriedades"via tradução, e dadas uma especificação SP e uma tradução τ que interprete SP, diz-se que uma especificação SP' *refina via a tradução* τ *a especificação* SP, em símbolos $SP \rightharpoonup_\tau SP'$, se $SP^\tau \rightsquigarrow SP'$. Assim, dada uma especificação $SP = \langle \Sigma, \Phi \rangle$ e uma tradução τ que interprete SP, tem-se que $SP \rightharpoonup_\tau SP'$ sse $SP' \models \tau(\Phi)$ (ver Lema 2.4.8).

Considere-se, a título de exemplo, a especificação do conjunto dos números naturais `NAT`, axiomatizada exclusivamente pelo axioma (∀`x,y:nat`).`s(x)=s(y)` ⇒ `x=y`, para `s` símbolo da função *sucessor.* Considere-se, também, a especificação `NATEQ` onde se especifica a função booleana "*equality test*"pelo seguinte conjunto de axiomas:

1. (∀`x:nat`).`eq(x,x)=true`;
2. (∀`x,y:nat`).`eq(x,y)=true` → `eq(y,x)=true`;
3. (∀`x,y,z:nat`).`eq(x,y)=true` ∧ `eq(y,z)=true` → `eq(x,z)=true`;
4. (∀`x,y:nat`).`eq(x,y)=true` → `eq(s(x),s(y))=true`;
5. (∀`x,y:nat`).`eq(s(x),s(y))=true` → `eq(x,y)=true`;

Considere-se agora a tradução τ definida por

$$\tau_{nat,bool}(\texttt{x:nat=y:nat}) = \{\texttt{eq(x:nat,y:nat)=true}\}$$

e por $\tau_{s,s'} = \emptyset$ noutros casos. Não é difícil de verificar que `NATEQ` interpreta `NAT` por τ tendo-se, portanto, que `NAT` $\rightharpoonup_\tau$ `NATEQ`, uma vez que `NATEQ` $\models$ `eq(s(x),s(y))=true` ⇒ `eq(x,y)=true` (cf. [Mad08, Exemplo 6.2]). Observe-se que neste exemplo se traduziu uma especificação axiomatizada em equações do género `nat` numa outra axiomatizada exclusivamente por equações do género `bool`. Este facto pode ser importante, por exemplo, quando se pretende encapsular géneros.

Relativamente a trabalhos futuros neste tópico, seria interessante numa primeira fase, estudar a integração deste tipo de refinamentos no processo de refinamento passo-a-passo apresentado nas Secções 2.4 e 3.4, nomeadamente a sua combinação com os refinamentos via morfismos de assinatura e a sua adaptação ao paradigma da observabilidade. Um primeiro passo neste sentido pode ser dado a partir do estudo da generalização do conceito de *refinamento via tradução* ao caso "misto"onde dados um morfismo $\sigma : \Sigma \to \Sigma'$ e uma $\Sigma - \Sigma'$-tradução τ que interprete SP, $SP \rightharpoonup_{\tau \triangleright \sigma} SP'$ sse

$SP^{\tau} \rightsquigarrow_{\sigma} SP'$. Um outro tópico interessante para trabalho futuro é o de estudar a *equivalência entre especificações* na perspectiva das traduções lógicas. Este trabalho pode ser feito com base no estudo da *equivalência de sistemas dedutivos* no sentido de [BP98].

6.1.2 Estudo Comparativo entre as Relações $\approx_{Obs,In}$ e $\approx^{\Gamma}_{Obs}$

Estudaram-se neste trabalho, duas definições para o conceito de igualdade observacional entre elementos: a *igualdade observacional parcial* $\approx_{Obs,In}$ (Definição 3.1.16) dos trabalhos de *Bidoit-Hennicker*, e a relação de Γ-*igualdade observacional* $\approx^{\Gamma}_{Obs}$ (Definição 4.1.1) dos trabalhos de *Goguen-Roşu*. Em ambas as relações, consideram-se apenas os contextos representantes das "computações admissíveis", usando-se para este fim, na primeira relação, os contextos definidos por variáveis de *input* e na segunda, os contextos definidos por operações de Γ. Um interessante tópico a desenvolver, seria o de comparar estas duas escolhas, no sentido de estudar a possibilidade de definir subassinaturas Γ e conjuntos de géneros In em função de $\approx^{\Gamma}_{Obs}$ e $\approx_{Obs,In}$, por forma a definirem relações equivalentes.

Este trabalho parece possível, pela definição de morfismos de assinaturas σ e ϕ que enriqueçam de forma adequada a assinatura da especificação a verificar, de tal forma que $\approx_{Obs,In}\restriction_{\sigma} = \approx^{\Gamma}_{Obs}$ e $\approx^{\Gamma}_{Obs}\restriction_{\phi} = \approx_{Obs,In}$.

Índice

Bibliografia

[BBK95] G. Bernot, M. Bidoit, and T. Knapik. Observational specifications and the indistinguishability assumption. *Theor. Comput. Sci.*, 139(1–2):275–314, 1995.

[BBR98] N. Berreged, A. Bouhoula, and M. Rusinowitch. Observational proofs with critical contexts. In *Fundamental Approaches to Software Engineering, volume 1382 of LNCS, Springer*, pages 38–53. 1998.

[BCH99] M. Bidoit, M. Cengarle, and R. Hennicker. Proof systems for structured specifications and their refinements. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, pages 385–433. Springer, 1999.

[BG80] R. Burstall and J. Goguen. The semantics of clear, a specification language. In *Proceedings of the Abstract Software Specifications, 1979 Copenhagen Winter School*, pages 292–332, London, UK, 1980. Springer-Verlag.

[BH94] M. Bidoit and R. Hennicker. Proving behavioural theorems with standard first-order logic. In *Algebraic and Logic Programming*, pages 41–58, 1994.

[BH96] M. Bidoit and R. Hennicker. Behavioural theories and the proof of behavioural properties. *Theor. Comput. Sci.*, 165(1):3–55, 1996.

[BH98] M. Bidoit and R. Hennicker. Modular correctness proofs of behavioural implementations. *Acta Informatica*, 35(11):951–1005, 1998.

[BH99] M. Bidoit and R. Hennicker. Observer complete definitions are behaviourally coherent. In *Proc. OBJ/CafeOBJ/Maude Workshop at Formal Methods'99, Toulouse, France, Sep.*, pages 83–94. 1999. Preliminary long version available as Report LSV 99-4.

[BH08] D. Bjørner and M. Henson, editors. *Logics of Specification Languages*. Springer-Verlag Berlin Heidelberg, 2008.

[BHK03] M. Bidoit, R. Hennicker, and A. Kurz. Observational logic, constructor-based logic, and their duality. *Theor. Comput. Sci.*, 298(3):471–510, 2003.

[BHW95] M. Bidoit, R. Hennicker, and M. Wirsing. Behavioural and abstractor specifications. In *ESOP '94: Selected papers of ESOP '94, the 5th European symposium on Programming*, pages 149–186, Amsterdam, The Netherlands, The Netherlands, 1995. Elsevier Science Publishers B. V.

[Bil03] S. Bilaniuk. *A Problem Course in Mathematical Logic, Version 1.6*. 2003.

[BM04] M. Bidoit and P. Mosses. CASL *User Manual*. LNCS 2900 (IFIP Series). Springer, 2004. With chapters by T. Mossakowski, D. Sannella, and A. Tarlecki.

[Bor99] T. Borzyszkowski. Completeness of the logical system for structured specifications, lecture notes in computer science 1589, pages 16–30, 1999., 1999.

[Bou97] A. Bouhoula. Automated theorem proving by test set induction. *J. Symb. Comput.*, 23(1):47–77, 1997.

[BP89] W.J. Blok and D. Pigozzi. Algebraizable logics. *Memoirs of the American Mathematical Society*, 396, Amer. Math. Soc., Providence, 1989.

[BP92] W. J. Blok and D. Pigozzi. Algebraic semantics for universal Horn logic without equality. In *Universal algebra and quasigroup theory, Lect. Conf., Jadwisin/Pol. 1989, Res. Expo. Math. 19*, pages 1–56. 1992.

[BP98] W. Blok and D. Pigozzi. Abstract algebraic logic and the deduction theorem. 1998.

[BR95] A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *J. Autom. Reasoning*, 14(2):189–235, 1995.

[BR00] S. Buss and G. Roşu. Incompleteness of behavioral logics. In *Reichel, Horst (ed.), CMCS 2000. Coalgebraic methods in computer science, Berlin, Germany, March 25-26, 2000. Amsterdam: Elsevier, Electronic Notes in Theor. Comp. Sci. 33, 19 p., electronic only* . 2000.

[BR02] A. Bouhoula and M. Rusinowitch. Observational proofs by rewriting. *Theor. Comput. Sci.*, 275(1-2):675–698, 2002.

[BR03] W. Blok and J. Rebagliato. Algebraic semantics for deductive systems. *Studia Logica*, 74(1-2):153–180, 2003.

[BS81] S. Burris and H. P. Sankappanavar. *A course in universal algebra.* Graduate Texts in Mathematics, Vol. 78. New York - Heidelberg Berlin: Springer-Verlag., 1981.

[BST08] M. Bidoit, D. Sannella, and A. Tarlecki. Observational interpretation of casl specifications. *Mathematical Structures in Computer Science*, 2008.

[CM08] D. Cansell and D. Méry. The event-b modelling method: Concepts and case studies. In Dines Bjørner and Martin Henson, editors, *Logics of Specification Languages*, pages 47–144. Springer, 2008.

[Der05] N. Dershowitz. Term rewriting systems by "terese"(marc bezem, jan willem klop, and roel de vrijer, eds.). *Theory Pract. Log. Program.*, 5(3):395–399, 2005.

[DF00] R. Diaconescu and K. Futatsugi. Behavioural coherence in object-oriented algebraic specification. *J. UCS*, 6(1):74–96, 2000.

[DF02] R. Diaconescu and K. Futatsugi. Logical foundations of CafeOBJ. *Theor. Comput. Sci.*, 285(2):289–318, 2002.

[DJ90] N. Dershowitz and J. Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 243–320. 1990.

[DSW94] M. Davis, M. Sigal, and E. Weyuker. *Computability, complexity, and languages (2nd ed.): fundamentals of theoretical computer science.* Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[DW02] K. Denecke and S. L. Wismath. *Universal algebra and applications in theoretical computer science.* Boca Raton, Florida: Chapman & Hall/CRC, 2002.

[EFH83] H. Ehrig, W. Fey, and H. Hansen. Act one - an algebraic specification language with two levels of semantics. In *ADT*, 1983.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification I.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1985.

[EM90] H. Ehrig and B. Mahr. *Fundamentals of algebraic specification 2: module specifications and constraints.* Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[End01] H. B. Enderton. *A mathematical introduction to logic, 2nd ed.* San Diego, CA: Harcourt/Academic Press, 2001.

[FL98] J. Fitzgerald and P. Larsen. *Moddelin Systems-Pratical Tools and Techniques in software Development.* Cambridge University Press, 1998.

[GB92] J. Goguen and R. Burstall. Institutions: abstract model theory for specification and programming. *J. ACM*, 39(1):95–146, 1992.

[GB99] M. Gaudel and G. Bernot. The role of formal specifications. In H-J.Kreowski E.Astesiano and B.Krieg-Bruckner, editors, *Algebraic Foundations of Systems Specification*, chapter Chapter I, pages 1–12. Springer-Verlag, 1999.

[GD94a] J. Goguen and R. Diaconescu. An oxford survey of order sorted algebra. *Mathematical Structures in Computer Science*, 4(3):363–392, 1994.

[GD94b] J. Goguen and R. Diaconescu. Towards an algebraic semantics for the object paradigm. In *RECENT trends in data type specification: workshop on specification of abstract data types: COMPASS: selected papers*, number 785. Springer Verlag, Berlin, Germany, 1994.

[GLR00a] J. Goguen, K. Lin, and G. Roşu. Behavioral and coinductive rewriting. In *Futatsugi, Kokichi, The 3rd international workshop on rewriting logic and its applications, RWLW. Kanazawa City Cultural Hall, Kanazawa, Japan, September 18-20, 2000. Amsterdam: Elsevier, Electronic Notes in Theoretical Computer Science. 36, 22 p., electronic only* . 2000.

[GLR00b] J. Goguen, K. Lin, and G. Rosu. Circular coinductive rewriting. In *Proceedings, Automated Software Engineering '00 (Grenoble France), IEEE Press*, pages 123–131. 2000.

[GLR02] A. Goguen, K. Lin, and G. Rosu. Conditional circular coinductive rewriting with case analysis. In *WADT*, pages 216–232, 2002.

[GM96a] J. Goguen and G. Malcolm. *Algebraic semantics of imperative programs.* MIT Press Series in the Foundations of Computing. Cambridge, 1996.

[GM96b] J. Goguen and G. Malcolm. Extended abstract of a hidden agenda. *Proceedings, Conference on Intelligent Systems: a Semiotic Perspective, National Institute of Standards and Technology*, 1996.

[GM99] J. Goguen and G. Malcolm. Hidden coinduction: Behavioural correctness proofs for objects. *Math. Struct. Comput. Sci.*, 9(3):287–319, 1999.

[GM00] J. Goguen and G. Malcolm. A hidden agenda. *Theor. Comput. Sci.*, 245(1):55–101, 2000.

[Gog99] J. Goguen. Hidden algebra for software engineering. In *Proceedings Combinatorics, Computation and Logic*, volume 21, pages 35–59, Auckland, New Zealand, 1999. Springer Verlag.

[GR99a] J. Goguen and G. Roşu. Hiding more of hidden algebra. In *Wing, Jeannette M. et al. (ed.), FM '99. Formal methods. World congress on Formal methods in the development of computing systems. Toulouse, France, September 20-24. Proceedings.* 1999.

[GR99b] J. Goguen and G. Rosu. A protocol for distributed cooperative work, in proceedings, workshop on distributed systems, 1999 (iasi, romania) electronic lecture notes in computer science, volume 28., 1999.

[Grä79] G. Grätzer. *Universal algebra. 2nd ed.* New York, Heidelberg, Berlin: Springer-Verlag, 1979.

[Hen90] R. Hennicker. Context induction: a proof principle for behavioural abstractions. In *DISCO '90: Proceedings of the International Symposium on Design and Implementation of Symbolic Computation Systems*, pages 101–110, London, UK, 1990. Springer-Verlag.

[Hen97] R. Hennicker. Structural specifications with behavioural operators: semantics, proof methods and applications, 1997. Habilitationsschrift, Institut für Informatik, Ludwig-Maximilians-Universität München.

[HN94] R. Hennicker and F. Nickl. A behavioural algebraic framework for modular system design with reuse. In *Selected papers from the 9th Workshop on Specification of Abstract Data Types Joint with the 4th COMPASS Workshop on Recent Trends in Data Type Specification*, pages 220–234, London, UK, 1994. Springer-Verlag.

[HR87] Jr. H. Rogers. *Theory of recursive functions and effective computability.* MIT Press, Cambridge, MA, USA, 1987.

[HUM01] J. E. Hopcroft, J. D. Ullman, and R. Motwani. *Introduction to automata theory, languages, and computation. 2nd ed.* Reading, MA: Addison-Wesley, 2001.

[HWB97] R. Hennicker, M. Wirsing, and M. Bidoit. Proof systems for structured specifications with observability operators. *Theor. Comput. Sci.*, 173(2):393–443, 1997.

[LEW00] J. Loeckx, H.-D. Ehrich, and M. Wolf. Algebraic specification of abstract data types. In *Handbook of logic in computer science, Vol. 5*, Oxford Sci. Publ., pages 217–316. Oxford Univ. Press, New York, 2000.

[Mad08] A. Madeira. Observational refinement process. volume REFINE 2008-International Refinement Workshop, Turku Finland. Electronic Notes in Theoretical Computer Science (To appear). Elsevier, 2008.

[Mar00] F. Martins. *Programação Orientada aos Objectos em Java 2.* FCA - Editora de Informática, 2ª edition, 2000.

[Mar04] M. A. Martins. *Behavioral Reasoning in generalized hidden Logics.* PhD thesis, University of Lisbon, Faculdade de Ciências, 2004.

[Mar06] M. A. Martins. Behavioral institutions and refinements in generalized hidden logics. *Journal of Universal Computer Science*, 12(8):1020–1049, 2006.

[Mar07] M. A. Martins. Closure properties for the class of behavioral models. *Theor. Comput. Sci.*, 379(1-2):53–83, 2007.

[Mar08] M. A. Martins. On the behavioral equivalence between k-data structures. *Comput. J.*, 51(2):181–191, 2008.

[Mat98] M. Matsumoto. Verification methods for behavioural specifications. Master's thesis, Japan Advanced institut fo Science and technology, 1998.

[MHST08] T. Mossakowski, A. Haxthausen, D. Sannella, and A. Tarlecki. CASL—the common algebraic specification language. In Dines Bjørner and Martin Henson, editors, *Logics of Specification Languages*, pages 241–298. Springer, 2008.

[MP07] M. Martins and D. Pigozzi. Behavioural reasoning for conditional equations. *Mathematical. Structures in Comp. Sci.*, 17(5):1075–1113, 2007.

[MT92] K. Meinke and J. Tucker. Universal algebra. pages 189–368, 1992.

[NSF] A. Nakagawa, T. Sawada, and K. Futatsugi. Cafeobj user's manual ver. 1.4.

[Pad98] P. Padawitz. Swinging data types: The dialectic between actions and constructors. Available at `http://ls5-www.cs.uni-dortmund.de/~peter/Swing.ps.gz`, 1998.

[Rei85] H. Reichel. Behavioural validity of conditional equations in abstract data types. In *Contributions to general algebra 3, Proc. Conf., Vienna 1984*, pages 301–324. 1985.

[RG00] G. Roşu and J. Goguen. Hidden congruent deduction. In *Caferra, Ricardo (ed.) et al., Automated deduction in classical and non-classical logics. Selected papers. Berlin: Springer. Lect. Notes Comput. Sci. 1761*, pages 251–266. 2000.

[RG01] G. Roşu and J. Goguen. Circular coinduction. International Joint Conference on Automated Reasoning (IJCAR'01), Sienna, 2001. Available at `http://www-cse.ucsd.edu/users/goguen/pps/ccoind.ps`.

[Ros99] G. Rosu. Behavioral coinductive rewriting, proceedings, obj/cafeobj/maude workshop, toulouse, france, 20th and 22nd september 1999., 1999.

[Roş00] G. Roşu. *Hidden Logic*. PhD thesis, University of California, San Diego, 2000.

[Rum91] J. Rumbaugh. *Object-oriented modeling and design*. Prentice-Hall International Editions, 1991.

[San00] D. Sannella. Algebraic specification and program development by stepwise refinement. (Extended abstract). In *Bossi, Annalisa (ed.), Logic-based program synthesis and transformation. 9th international workshop, LOPSTR '99. Venice, Italy, September 22-24, 1999. Selected papers. Berlin: Springer. Lect. Notes Comput. Sci. 1817*, pages 1–9. 2000.

[Sch92] O. Schoett. Two impossibility theorems on behaviour specification of abstract data types. *Acta Inf.*, 29(6/7):595–621, 1992.

[Ser93] C. Sernadas. *Introdução à teoria da computação*. Editoral Presença, 1993.

[Sim08] S. Simpson. *Foundations of Mathematics. Lecture Notes*. 2008.

[Sip96] M. Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1996.

[ST86] D. Sannella and A. Tarlecki. Extended ML: an institution independent framework for formal program development. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, *Proceedings, Summer Workshop on Category Theory and Computer Programming*, pages 364–389. Springer, 1986.

[ST97] D. Sannella and A. Tarlecki. Essential concepts of algebraic specification and program development. *Formal Asp. Comput.*, 9(3):229–269, 1997.

[STar] D. Sannella and A. Tarlecki. *Foundations of Algebraic Specifications and Formal Program Development*. Cambridge University Press, To appear.

[SW83] D. Sannella and M. Wirsing. A kernel language for algebraic specification and implementation - extended abstract. In *FCT*, pages 413–427, 1983.

[Tar03] A. Tarlecki. Abstract specification theory: an overview. In M. Broy, editor, *Models, Algebras, and Logics of Engineering Software, Volume 191 of NATO Science Series*. IOS Press, 2003.

[WD96] J. Woodcock and J. Davies. *Using Z: specification, refinement, and proof.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[Wir90] M. Wirsing. Algebraic specification. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)(Jan van Leeuwen edt.)*, pages 675–788. 1990.

[Wol87] U. Wolter. The power of behavioural validity. *TU Magdeburg,sektion Mathematik*, 1987.